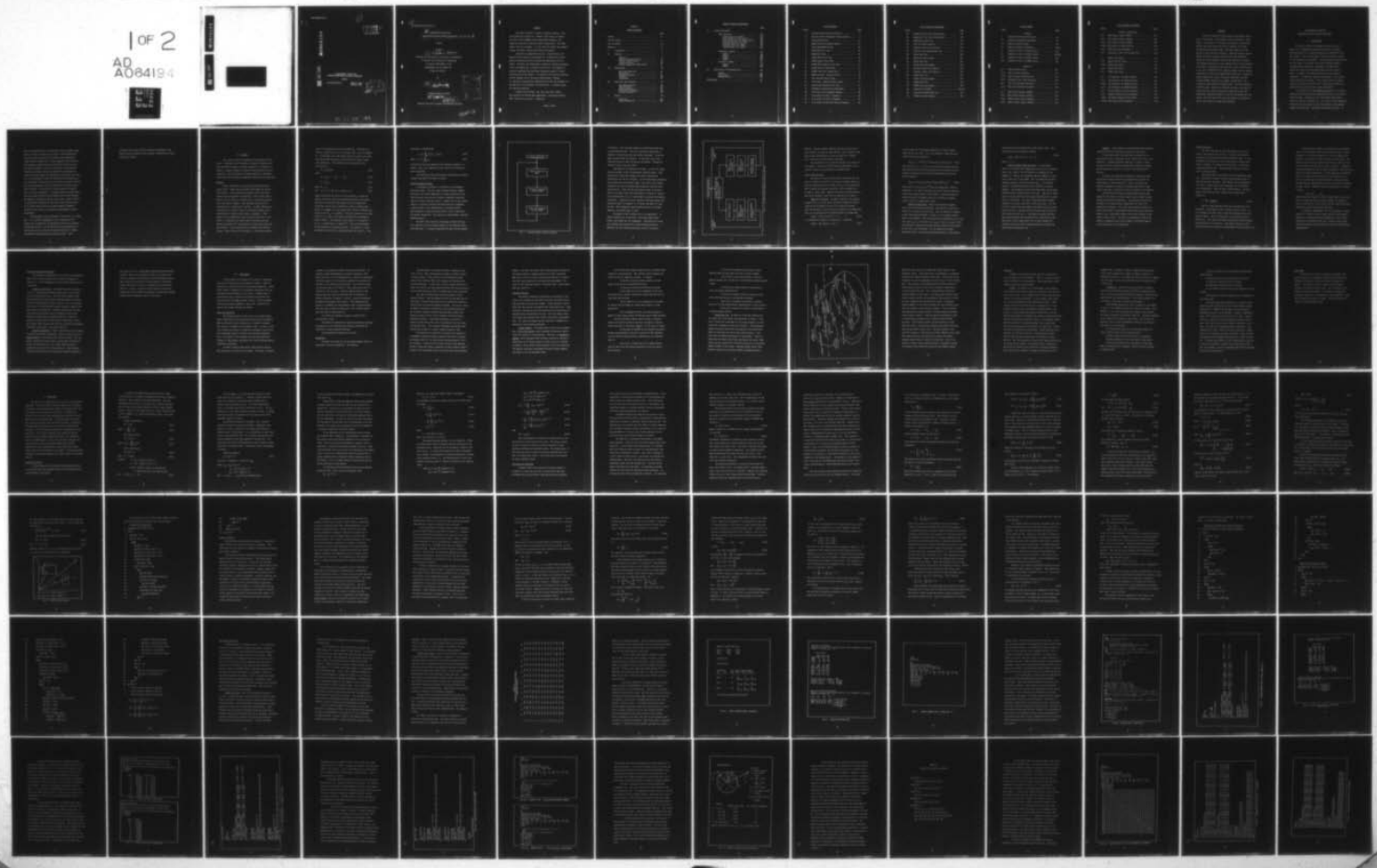
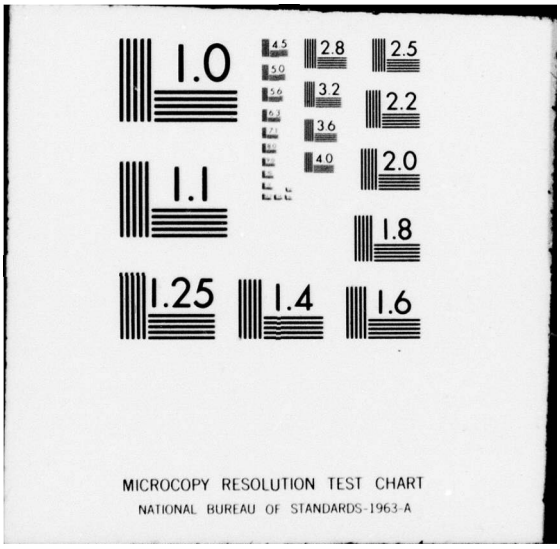
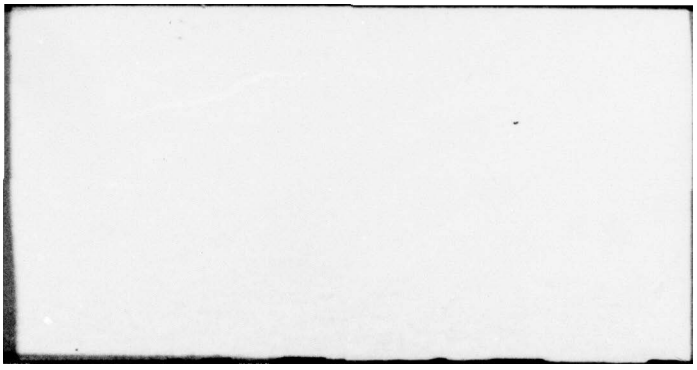
AD-A064 194 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 5/8
A DEVELOPMENT SYSTEM FOR MICROPROCESSOR BASED PATTERN RECOGNIZE--ETC(U)
DEC 78 J R LEARY

UNCLASSIFIED AFIT/GCS/EE/78-12-VOL-1 NL

1 OF 2
AD
A064194




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A



AFIT/GCS/EE/78-12

(1)

LEVEL II

AD A064194

DDC FILE COPY

A DEVELOPMENT SYSTEM FOR MICROPROCESSOR BASED PATTERN RECOGNIZERS

THESIS

AFIT/GCS/EE/78-12

John R. Leary
Captain, USAF

(14) AFIT/GCS/EE/78-12-Vol-1

(6) A DEVELOPMENT SYSTEM FOR
MICROPROCESSOR BASED PATTERN RECOGNIZERS, Volume I.

THESIS

VOLUME I

(9) Master's thesis,

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

(10) John R. Leary, [redacted].
Captain USAF

Graduate Computer Systems

(11) December 1978

(12) 176p.

Approved for public release; distribution unlimited

012225

Preface

This thesis presents a system of computer programs. They are designed for student use. However, their design is modular, the code is ANSI FORTRAN and the common 8080 assembler. This design was selected to make the system transportable. Over 4000 source lines are included. If a user does not require the complete system, individual routines may easily be extracted.

Some notes of appreciation are due. Charlie Dutra, Tom Gabrielle, Gene Mechler, and Professor V. O. McBrien all participated in educating me and in creating the opportunity for this thesis. My typist, Ms. Nancy Myers, produced an amazing transformation in the manuscript in almost no time at all. The members of my thesis committee have graciously endured my moments of confusion and given solid support. I am thankful for Professor Richard's careful comments and Dr. Hartrum's understanding. Without Dr. Kabrisky's perspicacious underwriting not even the statement of my objectives in this bottom line would exist. I sincerely thank all who have helped me.

A special note follows: MJ, Jack, Amy, Moira, Nancy - your patience with me has been magnificent. You have my promise that 'the best is yet to be.' Thank you.

John R. Leary

Volume I

Table of Contents

Table of Contents (Continued)

List of Figures

List of Figures (Continued)

List of Tables

List of Tables (Continued)

Abstract

A tool for developing microprocessor based pattern recognizers is presented. A two segment system of programs is implemented. One segment is a subsystem consisting of a generalized pattern classifier program and utility routines for an INTEL SBC 80/20 microprocessor system. The other segment is a subsystem of four interactive programs. These four programs support feature selection, pattern class definition and performance evaluation using procedures fitted to the classifier algorithm. This subsystem operates on a user supplied file of feature vectors. It produces a class defining structure for use by the classifier. It can use a TEKTRONIX 4014 for graphics support and will operate interactively within the CDC 6600 Intercom partition. Structured design, modular code, buffer allocation algorithms, and ANSI standard FORTRAN code make this segment transportable. The classifier segment requires an 8080 system. Less than 256 bytes of ROM are used. Data buffer locations and sizes, the number of classes and the number of features are specified by the user. Experiments produced estimates of classifier performance for this system. An error rate of less than ten percent is reported for one 26 class character recognition experiment.

A DEVELOPMENT SYSTEM FOR MICROPROCESSOR BASED PATTERN RECOGNIZERS

I. Introduction

This thesis presents a development system for use as a design tool in implementing experimental pattern recognizers. Some characteristics of pattern recognizers are described in the next chapter. The art of designing a pattern recognition system is also discussed in that chapter. The development system produced for this thesis is discussed in the three following chapters. In chapter three, functional requirements are established. Chapter four defines the algorithms upon which this system is based. In chapter five, the design and use of the system is documented. Two questions remain to be addressed. Their answers justify the above discussion. First, of what value are pattern recognition systems to the Air Force? And second, how does this system relate to such Air Force pattern recognizers?

In a recent issue of Air University Review, Dr. Paul Namin explores the military need for Identification Friend, Foe, Neutral (IFFN) systems. He makes the point that without such systems there is a serious limitation, i.e., the rule of visual engagement, which restricts the degree to which the potential of any weapons system can be realized. An anecdote illustrates his point. It

tells of the destruction of a multimillion dollar weapons system while its pilot is unaware of any threat. Namin hypothesizes that this might occur because of marginal enemy advantage in target detection capability. He then suggests that solutions to the technology problem posed by IFFN need not necessarily seek new sensor phenomena. Rather, he holds that a more effective integration of sensor data may be produced by enhancements to signal processing systems and "shrinkage in device cost and size." This is the synergistic effect of "getting more performance out of a collection of data than any one of them can provide." (Ref 9) This may be the general military application for pattern recognition systems. At each node of a complex network of sensors may lie a pattern recognizer. It reduces volumes of higher level data into simple classification statements which funnel through the network as command and control status items. Namin's IFFN is "a technological challenge for the '80s." Classification inputs to C^4 status networks begin with simple pattern recognizers applied to small pieces of the complex electromagnetic warfare environment.

The development system presented in this thesis is a simple one. It is primarily pedagogical, and is intended for AFIT student use in exploration of experimental solutions to specific recognition problems. But the concept and the configuration of this system are also aimed at the practical problem of cheaply implementing prototype pattern recognizer systems. Such

prototypes may provide sufficient empirical knowledge of key sensor data environments for the ultimate implementation of high reliability systems.

II. Concepts

This chapter presents the theoretical foundations for the thesis. Following a brief statement of notation conventions and some definitions, design of pattern recognizers is discussed in general. Concepts relevant to classification algorithms are next presented. Then the selection of pattern features is discussed. Finally two types of pattern recognition applications are described.

Notation

Several definitions and notation conventions make this report easier. Assume that any pattern environment may consist of N patterns. These patterns may separate into I sets whose members share some degree of commonality. Each of these sets of patterns will be known as a pattern class. An arbitrary pattern class may contain L members. Any individual pattern may be represented by J characteristic features. If these features are considered as an ordered J-tuple, an individual pattern can be represented by a feature vector having J components. These vectors will be referenced as 1XJ row matrices when this is convenient. A population of feature vectors collected from the pattern environment will be described as a data base or data set, and denoted Ω. This collection can be separated into disjoint classes. Each of these will be denoted ω. An arbitrary feature

vector in the population Ω will be denoted F_n. Similarly, an arbitrary feature vector in an arbitrary class ω_i will be denoted F_ℓ. A consistent use of the single subscripts n and ℓ will overcome any possible ambiguity in specification of feature vector class membership. These definitions are summarized in the notation below.

$$\Omega = \{F_n \mid 1 \leq n \leq N\} \tag{2-1}$$

where

$$F_n = (f_1, \ldots f_j, \ldots f_J) \tag{2-2}$$

$$\Omega = \bigcup_{i=1}^{I} \omega_i \tag{2-3}$$

where $F_\ell \in \omega_i$ (2-4)

and $\omega_i \cap \omega_k = \phi$ for all i, k when $i \neq k$ (2-5)

Symbol conventions are implicit in this notation. Vector components and scalar values are represented by lower case letters. Subscripts are used only when needed to clarify significant differences and not used to establish a trail of relationships. Thus F_ℓ is a member of ω_i and context will suffice to identify the vector of which f_j is a component. With the exception of the index limits N, I, J, and L, only vectors and matrices are denoted by capital letters. The transpose of the usual I x J row matrix F to a J x I column matrix is denoted F^T. There is one exception to the convention for denoting matrices. The symbol Σ_i is used to denote the within-class covariance matrix for class ω_i. This

covariance is estimated by:

$$\Sigma_i = \frac{1}{L} \sum_{\ell=1}^{L} (F_\ell^T F_\ell - P_i^T P_i) \qquad (2\text{-}6)$$

where $$P_i = \frac{1}{L} \sum_{\ell=1}^{L} F_\ell. \qquad (2\text{-}7)$$

In equation (2-6), the notation $F_\ell^T F_\ell$ indicates a square J x J matrix. Also, $F_\ell F_\ell^T$ denotes the scalar which is the square of vector magnitude.

The definitions above make possible explanations of several concepts upon which this thesis is based.

Pattern Recognizer Design

To recognize a pattern is to perceive it as something previously known. With this simple statement Webster suggests what Kanal (Ref 23:701) emphasizes as a major evolution of the last few years: the design of a pattern recognition system has come to be highly iterative process. A major part of this design process is acquiring necessary and sufficient prior knowledge. A major problem in this design process is deciding exactly what knowledge is necessary and how much of that is sufficient for pattern recognition. This decision is made through a two-path modeling process.

Box (Ref 2:24) discusses a philosophy of model building. Fig 1 presents his three stage procedure to find adequate models from known data. In pattern recognition the data are the patterns

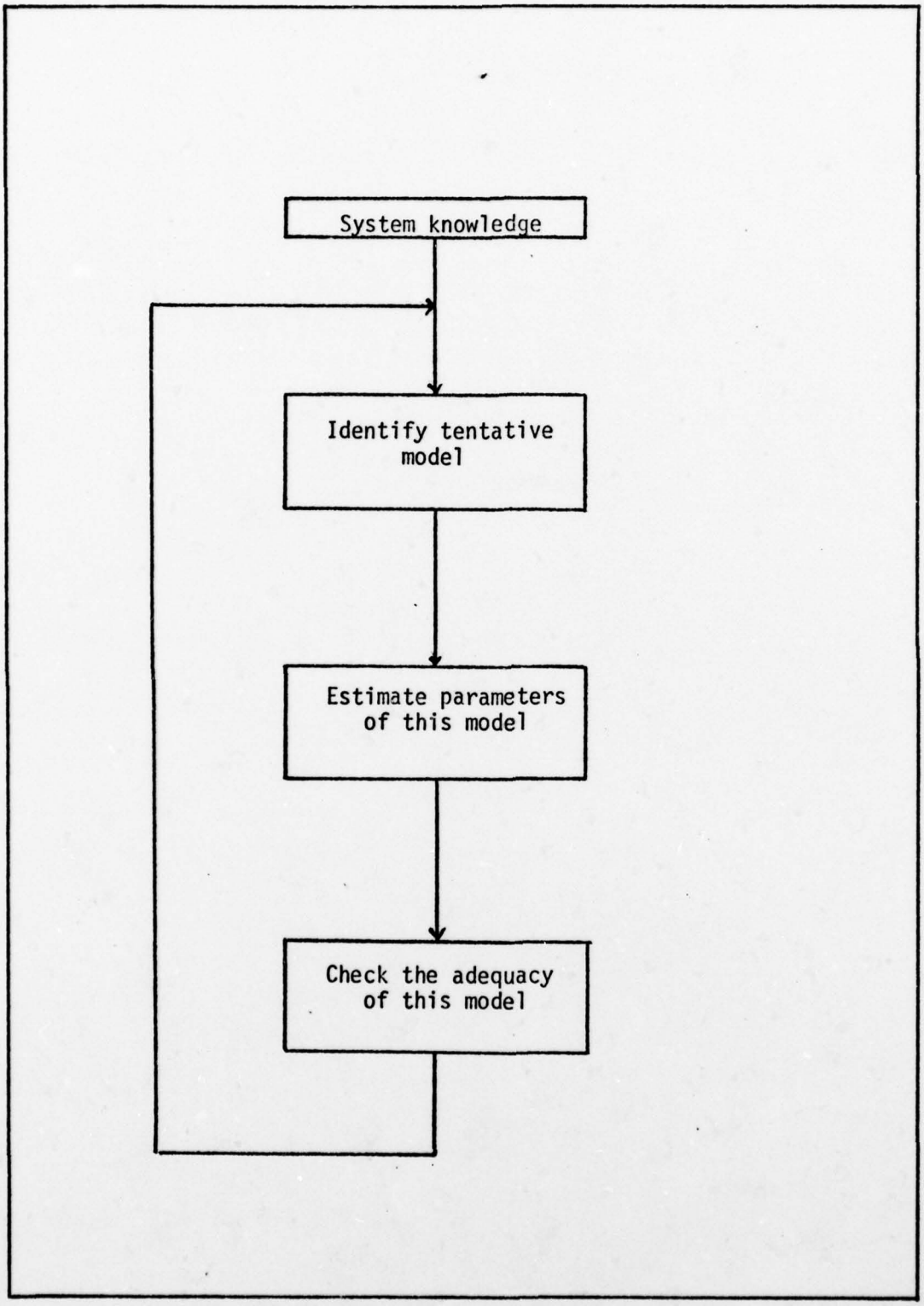


Fig. 1. Iterative Model Building Procedure

of interest. Here two paths produce a classification model and a representation model. These are respectively equivalent to Webster's present perception and previous knowledge. In the one path, features model the patterns. In the other path, class defining structures model the pattern environment. Through the former we come to know the latter.

Box explains his procedure as follows. In the first stage system knowledge is used to hypothesize tentative models. Here statistically inefficient methods are used because precise formulations are not yet available. In the second stage, parameters are estimated for the tentative model. Non-linear least squares procedures are used to estimate these parameters and then covariance matrices. After fitting the tentative model to observed data, in the third stage, the fitted model is checked in relation to the observations so as to reveal model inaccuracies and achieve improvement. Inspection of error functions indicates whether the entertained model is adequate, or if and how the model is to be revised. After diagnostic checks satisfy the user as to model adequacy, the derived model is used.

The appeal of Box's process lies in its generality. It applies equally well to each path. Fig 2 shows these paths. Clearly these paths are not independent. Production of an error rate requires both features and a class defining structure. Obviously the class defining structure is built in terms of

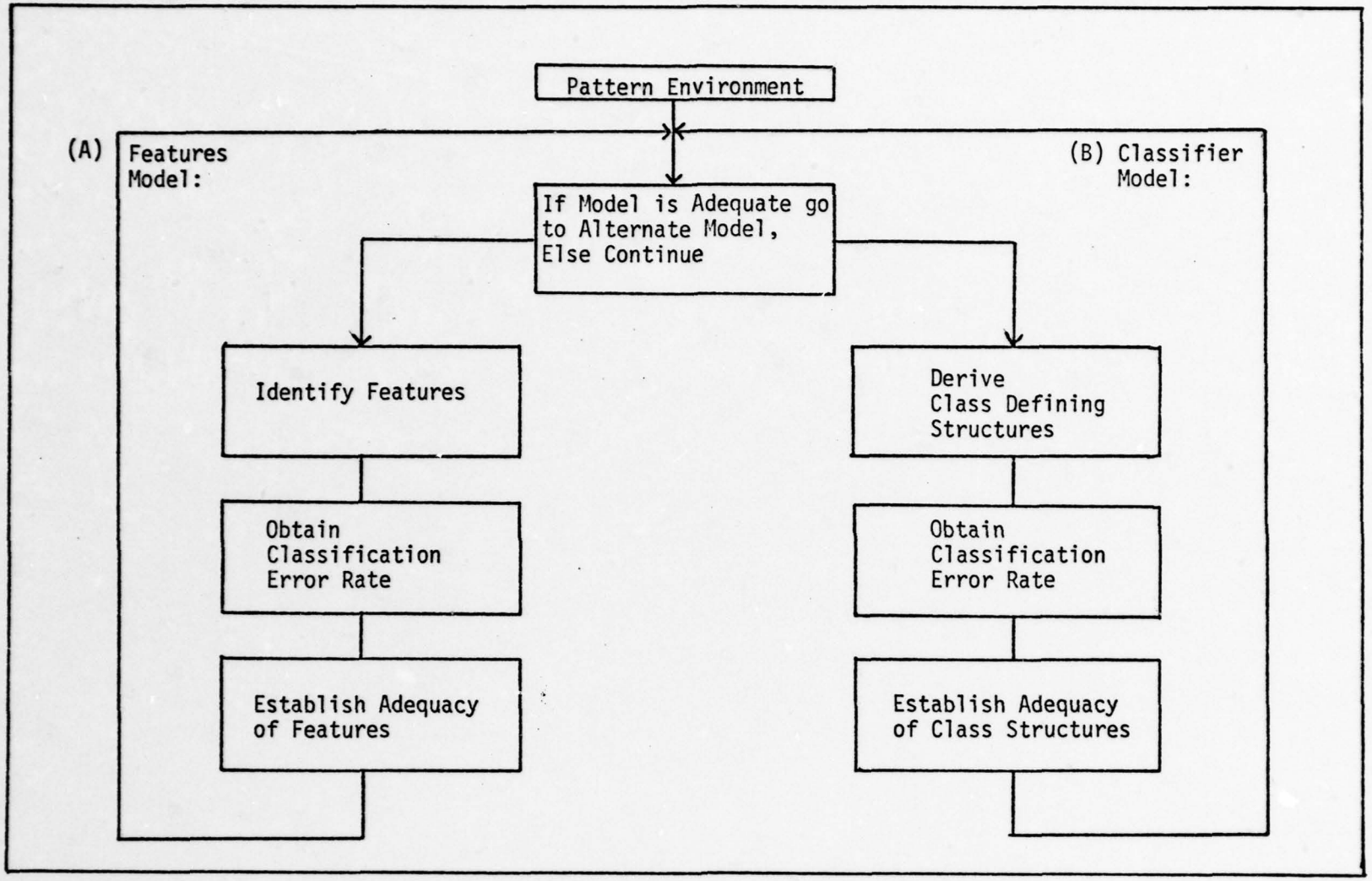


Fig. 2. Two Path Pattern Recognizer Design Iteration

features. However, feature identification does not end once a class defining structure has been derived. Nor is pattern recognizer design complete once an error rate has been validated. This is the point of this general discussion.

The two paths of Fig 2 lead into the next two sections of this chapter. They cover feature selection and pattern classification. Pattern classification is presented first.

Pattern Classification

Put simply, in terms of the notation stated earlier, the task of a pattern classifier is to assign an unknown pattern F'_n from an unknown data set Ω' to that class $\omega_i \subset \Omega$ with whose members F'_n shares the greatest similarity. This assignment can be made in several ways. Bayesian classifiers, minimum distance and nearest neighbor classifiers are germane to this thesis.

Bayesian Classifiers. In these classifiers the a priori probability of ω_i and the class-conditional probability density functions of the members of class ω_i are explicitly known. Decision functions $d_i(F_n)$ are used to establish class membership. That is, the probability of misclassification is minimum when

$$d_i(F_n) = p(F_n|\omega_i)\, P_r(\omega_i),\ i = 1, \ldots I \tag{2-8}$$

is a maximum with respect to a choice of i. Therefore

$$d_k(F_n) = \max_i \{d_i(F_n)\} \rightarrow F_n \varepsilon\, \omega_k \tag{2-9}$$

In this expression the a priori probability is often assumed identical for each class. It is also common to assume the multivariate normal density which is

$$p(F_n|\omega_i) = (2\pi^{J/2}|\Sigma_i|^{\frac{1}{2}})^{-1}\exp(-\tfrac{1}{2}(F_n-P_i)\Sigma_i^{-1}(F_n-P_i)^T) \quad (2\text{-}10)$$

The symbols F_n, P_i, Σ_i, J and ω_i are all used as earlier defined. Using equation (2-9) a decision function can be written using the monotonic log function to simplify the exponential form of the Gaussian density.

$$d_i(F_n) = \ln[P_r(\omega_i)]-\tfrac{1}{2}\ln|\Sigma_i|-\tfrac{1}{2}(F_n-P_i)\Sigma_i^{-1}(F_n-P_i)^T \quad (2\text{-}11)$$

(Dividing all $p(F_n/\omega_i)$ by $2\pi^{J/2}$ does not change their relative magnitudes.) In a Bayes classifier, the set of decision functions relates the unknown pattern to all classes. The maximum decision function provides the index of the class to which the unknown feature vector is assigned (Ref 13:13).

Minimum Distance Rules. Many classification procedures can be said to follow this technique. The simplest of them first establish a prototype for each class. Then the unknown is assigned to that class whose prototype is closest, in a Euclidean distance sense, to the unknown. This rule requires two assumptions. One is that in F_ℓ and $F_{\ell+1} \in \omega_i$, the vector $(F_\ell - F_{\ell+1})$ is also in ω_i (Ref 12:11). This concept is required to justify the usual choice of the centroid of the class as its prototype. It also supports the second assumption which is that similarity between pattern is consistently

reflected by the Euclidean metric on the feature space. This rule can be concisely stated as follows:

$$d_k(F_n) = \min_i \{|F_n - P_i|\} \rightarrow F_n \in \omega_i \qquad (2\text{-}12)$$

where $1 \leq i \leq I$.

Nearest Neighbor (NN) Classifiers. Fix and Hodges (Ref 11) are credited with suggesting a variant of this classification rule. Again a set of distances are computed for the unknown F_n. However, the assumption that the members of a class form a convex set is not needed. This is because the measured distances relate F_n to each F_ℓ within each ω_i. The unknown pattern is assigned to the class which contains its nearest neighbor. The assumption that the Euclidean metric consistently reflects pattern similarity must still exist. The rule is robust since it can be sensitive to any actual distribution of F_ℓ given that Ω is sufficient. If a vote is taken among the K nearest neighbors of F_n then a K-NN rule is said to be used. The risk of error in this latter rule tends to the Bayes risk as K and N tend to infinity. Das Gupta (Ref 9:15) notes that NN rules are also related to rules based on estimates of density functions. The obvious problems with the NN rule are a sensitivity to bad data points, and a computational cost for data storage and execution time which tends to become excessive as the NN risk tends toward the Bayesian risk.

Comments. Three comments on classification rules establish a perspective for the algorithms developed in this thesis.

(1) Das Gupta (Ref 9:15) notes that the usefulness of a classification rule is determined by its simplicity as well as its robustness. Although conceptual simplicity is useful in that a rule may be easily understood, computational simplicity produces the efficiency which permits a rule to be used effectively in practice.

(2) There are complicated treatments of indecision zones and tolerance regions which may be asymptotically optimal for large numbers of classes (Ref 9:13). These may justify the simplistic approach of covering the feature space with as many "tight" subclasses as possible in order to optimize classification.

(3) Chen (Ref 4:6) notes that experimental results have established that there is always a small subset of good learning samples which dominate performance. This possible insensitivity to sample size of good quality neighborhoods can lead to an experimental procedure. In it, one uses analytical intuition to uncover the kernel of good-neighbor patterns which may define the optimal class. Undesirable samples can be said to belong to the "husk" of such a class. The idea is an outgrowth of that of the edited or condensed NN rule which attempts to eliminate samples on the wrong side of class boundaries.

Feature Selection

The term "identify" was used deliberately in the first block of the features path in Fig 2. It covers extraction of measurements which characterize digitized pattern data. It also encompasses the selection of the minimum subset of these values which is adequate for acceptable classification. Extraction is a problem dependent task. The more general question of selection is addressed below.

The problem here is essentially one of computational benefit. The number of features extracted from the pattern data is often deliberately too great. (See Chapter 4 under benchmarks.) This leaves a need to reduce the measurement set to one whose size is manageable. There are many possible subsets. The total number to be evaluated when j features are selected from J features is

$$T = \binom{J}{j} = \frac{J!}{j!(J-j)!} \qquad (2\text{-}13)$$

There are many techniques which have been applied to this evaluation. The problem is one of choosing a better subset. It is an accepted fact that there is only one guaranteed way to find the best subset. Cover has shown this to be exhaustive search (Ref 8:117). Jain reports that added features may actually degrade the performance of a classifier. Thus subset selection is motivated by more than an interest in computational efficiency (Ref 21:1).

Subset selection methods are basically search procedures. There is basic agreement that the best control on such a search procedure is to estimate probability of error by computing the empirical error rate on a large test data set (Ref 34:72). The simplest subset selection algorithms establish a figure of merit for each feature and then pick the best n features. Sequential ordering processes are used to reduce computation. Chen (Ref 3:89) notes that dynamic programming is a good technique for sequential search. He states that the search for one best feature at a time is computationally the most efficient. Stearns describes the bias that may unintentionally derive from previous selections in such a search. Sequential searches produce nests of subsets in which

$$S_1 \subset S_2 \subset \ldots S_n.$$

Features that are "powerful" in early stages remain in the final set even though they may no longer be needed. He suggests a "plus m, take away n" search to avoid the fact that the two best features may not be the best pair (Ref 34).

In summary, computational cost is a key factor in subset selection. The most critical element of any search procedure appears to be evaluation of error probability. This is best estimated by an empirical error rate. Finally, while nested selection procedures may bias results, they offer efficiency of implementation.

Pattern Recognition Applications

The algorithms implemented for this thesis are evaluated in terms of two differing applications of pattern recognizers in Chapter IV. A brief background on these different applications is given below.

Character Recognizers. Considerable work has been done at AFIT in investigating techniques which apply to the recognition of two-dimensional data. In these efforts features have been extracted from various digital representations of pictures using the two-dimensional Fourier transform. This is consistent with the work of Kabrisky whose research produced a model of the human visual system (Ref 22). Tallman's dissertation indicates that hardprinted characters can be recognized by use of low frequency filtered Fourier components (Ref 35). Efforts by Sponaugle to generalize this work towards recognition of multi-font typeset letter data are the basis for test data and benchmark comparisons given later in this report (Ref 33).

Waveform Recognizers. Signal classification can use pattern recognition techniques to advantage. Feucht's recent article in Computer Design is motivated by this fact (Ref 10:68). Hall and Bouvier produced AFIT theses dealing successfully with waveform pattern recognizers (Refs 14, 1). Radar signature pattern recognizers are found in Air Force operations. The classifier algorithm implemented for this thesis was originally designed by

the author for use in a Space Object Identification application (Ref 25). Many of the procedures present in this thesis are eclectic outgrowths of the synergy of that development project. These range from the concept of biased samples to which Chen attests (Ref 4:60) to the use of asymmetric class boundaries (Ref 32). Finally, a sample of radar signatures was used by Kulchak (Ref 24) to produce the Frequency of Binary Words (FOBW) feature vectors referenced later in this report.

III. Requirements

In this chapter the structure of the thesis is developed. The goals and objectives of the project are stated. These are addressed in a short discussion of underlying assumptions. Thereafter follows a statement of the functional requirements for the development system produced in this effort. A bubble chart is presented and used to explain the concept of system data flow upon which this development system is based. A short statement of design and coding standards is then given. Selection of a name for the system concludes the chapter.

Goals and Objectives

The ultimate purpose of this thesis is to support experimental implementation of microprocessor based pattern recognizers. Meeting this goal requires production of a system of programs. This system is intended to be a designer's tool. As such, it aims to facilitate the process of recognizer development, and to drive that development towards a specific microprocessor implementation. The system is also intended to be used and modified by students as they develop, experiment with, and investigate pattern recognition algorithms.

In order to achieve these goals, three specific development objectives are stated for the system. Its design is required

to model a key recognizer element, the pattern classifier. To simplify student implementation of pattern recognizers, this model classifier is to be programmed for a specific microprocessor. The system design is also required to generalize the process of deriving a class defining data structure. The classifier bases its decisions upon this structure. Thus, system error-rate is a function of this structure. Effective generalization of this process makes the system an effective tool for designers of pattern recognizers in general. Finally, a series of benchmark performance measurements are required. These demonstrate the system as a framework for both potential users and experimenters. They also serve to qualify system worth. All of these requirements boil down to three specifics:

(1) Design and implement a pattern classifier for a microprocessor system.

(2) Design and implement the supporting functions necessary to generate the class defining data structure with which the classifier can make acceptable decisions.

(3) Experimentally demonstrate the above.

Assumptions

The worth of the goal set for the above becomes clear in a discussion of several assumptions. This follows.

Microprocessors are readily available, inexpensive, and small in size. Small microprocessor systems can become elements of large networks. These systems can be interfaced to large random access memories (RAM), disk storage, and high speed processing technology. In the light of Namin's concept which introduced this report, one should therefore assume that microprocessors must be addressed by any effort to upgrade sensor data processing.

The task of implementing a pattern recognizer crosses many disciplines. Data processing obstacles can be major ones to individuals otherwise highly qualified to analytically determine significantly discriminating pattern features. The task of tuning an optimal classifier or generating a class defining structure may similarly sidetrack would-be designers whose talents tend towards the more critical task of designing efficient feature extraction hardware. Given these postulates, the worth of a general purpose design tool with a pre-selected classifier algorithm becomes clear. This argument strengthens considerably when the would-be designer is a thesis student pressed by time.

Pre-selection of a simplistic classifier as an element of a recognizer system may provide a benefit aside from its economy. An optimum classifier can only optimize the processing of its input features. It may well be far more critical to the implementation of successful pattern recognizers to place limited "model-T" systems in the environment than to initially seek high performance

systems. The search for better input features becomes tedious and intractable without a computer yardstick for their evaluation. What better yardstick is there than the performance of a "model-T" classifier which operates in the actual data environment? The answer to the foregoing question is obviously moot. Future experiments may resolve it.

Required Functions

The specific objectives stated above were analyzed in the light of the concepts and techniques of pattern recognition which were presented in the previous chapter. Broad functional requirements were thus derived to accomplish the stated objectives. These functional requirements were then studied with data processing and software design considerations in mind. From this effort a data flow diagram was produced which reflects the overall system operation. This data flow diagram and the functions it embodies are described in the following paragraphs.

System Segments. The system should consist of two segments. One, a Classifier Segment, should implement the selected pattern classifier design in a microprocessor. The other, an Interpreter Segment, should implement those functions required to interpret a sample data set of feature vectors in such a way as is required to produce a class defining data structure fit for the classifier. The specific functional requirements for each of these segments are stated in the two paragraphs below.

(1) The Classifier Segment should consist of software which resides in a microprocessor. This software should implement the classifier and its supporting routines. It should:

(a) be able to assign unknown patterns to their proper classes with an acceptable error-rate.

(b) be able to record classification decisions.

(c) be coded so as to be independent of the locations and sizes of buffers required for feature data and for the class defining structure.

(d) be coded so as to be independent of the number of features and the number of classes which comprise a given application.

(e) be implemented within less than 256 bytes of memory to allow storage within one ROM data page of 100H locations.

(2) The Interpreter Segment should consist of software which can be used as readily as possible to produce a class defining structure for the former segment. In this sense it should

(a) be coded in FORTRAN using a top-down structured design, and conforming as closely as possible to ANSI standards so as to maximize intelligibility, modifiability, and transportability.

(b) be able to adjust the size of memory buffers used for data files and internal structures to fit the size of user resources.

(c) be able to generate and to refine a class defining data structure which fits the classifier segment.

(d) be able to select and evaluate a subset of pattern features for its capacity in discriminating between pattern classes.

(e) be able to support analytical evaluation of class and feature characteristics.

(f) be able to support efficient transfer of the class defining structure to microprocessor storage.

(g) be able to produce and document a simulated error-rate for the microprocessor implementation of the classifier.

(h) be able to operate in either an interactive or a batched computer process.

System Data Flow. An analysis of the data processed by the system led to the bubble chart presented in Figure 3. This chart reflects the requirement for two system segments and indicates their conceptual and physical interface. The Interpreter Segment processes feature data and generates class definitions. These two data types are the primary system currency. Class definitions are denoted prototypes for convenience. These are based upon the feature vector data provided to the system. These latter data are organized for efficient system use in the process labeled "CREATE" on the figure. Multiple feature vector files provide a capacity to store test samples, segregate patterns

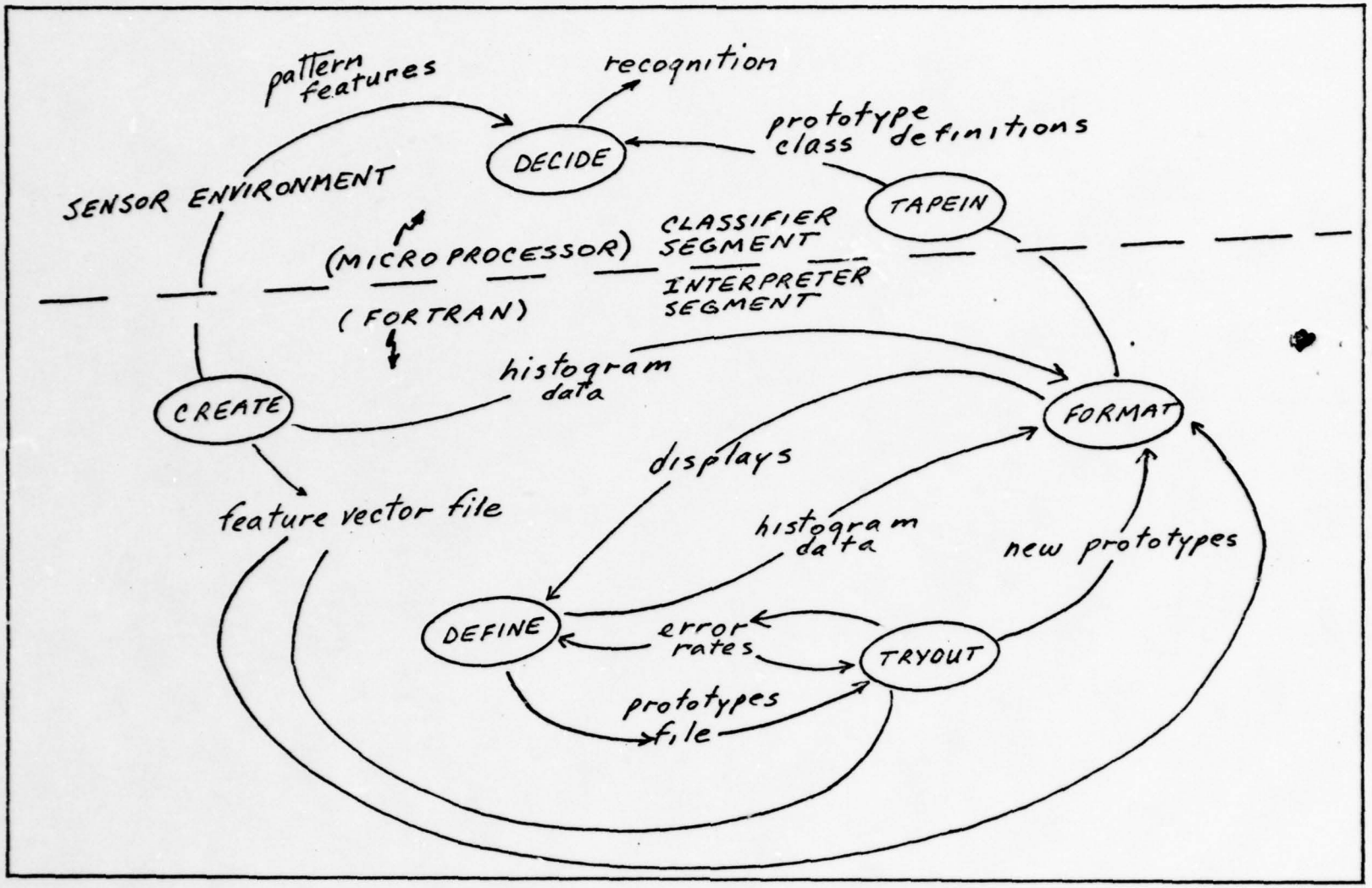


FIG 3. SYSTEM BUBBLE CHART - DATA FLOW

typical of data classes, and subset the overall data set into manageable pieces. Class definitions, or prototypes, are produced by the process labeled "DEFINE" on the chart. This process allows refinement of specific prototypes by selective use of input feature data. The capacity of the complete class defining structure to assign feature vectors to their proper classes is measured by a classification error-rate. This is documented by the process labeled "TRYOUT" on the figure. This same process supports selection of feature subsets, and evaluation of these subsets in terms of their respective classification error-rates. The process labeled "FORMAT" on the chart configures the class defining structure for transfer to the classifier segment. It also satisfies the requirement to support analysis of feature data by producing various graphic displays. These include three-dimensional plots of histogram data produced by the "CREATE" and the "DEFINE" processes. These displays reflect the distribution of values occurring within a given feature dimension both within the entire data set and within a given class. The basic process of the classifier segment is reflected by the label "DECIDE" on the figure. This process receives its input from the sensor environment through a process which is implicit on the chart. This is the process of feature vector generation which is assumed to operate in a parallel and controlling relation to the "DECIDE" process.

Standards

Standards are applied to ensure that the system which is produced meets general requirements. That is, it must be intelligible, modifiable, and transportable. These requirements affect software design and program coding.

Design. The expression of requirements in this chapter illustrates the key design standard to be applied to the development of this system. This standard requires that design decisions be made in a structured sequence. In this process, basic ideas are successively decomposed into subordinate concepts. These concepts are refined and the process is repeated until it has produced concrete tasks, specifications and definitions. The process is called structured design by IBM (Ref 20). Earlier, Niklaus Wirth termed it development by stepwise refinement (Ref 36). Applied to the design of computer software, the technique requires that the functions of a program solution first be specified. Then the data processed by each function are identified. Finally, functional relationships are determined. Program and data specifications are refined in parallel. Binding decisions about process logic and data representation are delayed as long as possible. Thus the advantages of various data formats become clear in contrast to one another. Processing paths are produced by choice and not forced by prior decision or arbitrary assumption. Wirth justifies his technique of stepwise refinement with the

argument that it produces a degree of modularity which greatly eases program adaptation to changes of purpose, function, or operating environment. This modularity therefore becomes a supporting requirement to ensure the modifiability and transportability of the system.

Programming. Adherence to American National Standards Institute (ANSI) FORTRAN standards facilitates transportability. Use of structured programming conventions enhances intelligibility, modifiability and transportability. Use of these standards and conventions is therefore a supporting requirement.

ANSI FORTRAN standards are clearly defined for CDC FORTRAN IV (Ref 7). This FORTRAN includes ANSI standard X3.9-1966. Since FORTRAN is a well-used and documented language, these standards are widely exceeded by off-the-shelf compilers. Therefore adherence to the standard often imposes a restriction. Some of the more important cases in which CDC FORTRAN IV should be restricted for this project are listed below.

(1) Input/output syntax will use the syntax READ (u,f) iolist or WRITE (u,f) iolist as defined by CDC.

(2) Data labels will be restricted to six characters.

(3) Data statements will not use implicit loop syntax.

(4) Hollerith constants will only appear in data statements or subroutine call statements, and will use the nH syntax as defined by CDC.

(5) Array references will be consistent with dimension specifications.

(6) Only sequential file access logic will be used.

(7) Subscript expressions will be avoided.

(8) Mixed mode expressions will be avoided.

(9) Non-standard system functions and subroutines will be avoided.

(10) Deviations from ANSI standards will be commented in the program code.

Structured programming conventions are guidelines which simplify program construction as much as they enhance program modifiability. FORTRAN does not admit such key structured programming constructs as the DO-WHILE. Moreover, FORTRAN provides a GOTO construct which must be used at times. However, inasmuch as possible structured programming technique will be used. When logic structures are complex, indentation will be used. The code will be segmented as much as possible. Each subroutine will have a single entry and a single exit. Module sizes will be limited to one page if possible. Logic flow will be sequential, with imbedded procedure calls, as much as possible. To ensure intelligibility of the program code, a ratio of at least one explanatory comment to each seven source lines will be maintained. Finally, meaningful names will be used wherever possible.(Ref 20:8-1).

System Name

Consistent with the last convention stated above, the name assigned to this development system should be descriptive. An 8080 microprocessor system is available to support this project. The system's classifier segment will be coded to operate on this microprocessor system. This classifier is defined in the following chapter. It references n dimensional rectangular regions in its assignment of class membership. These can be visualized as boxes in n-space. For these reasons, the system is called the BOX80 system.

IV. Algorithms

The design of the BOX80 system rests upon its classification algorithm. A specialized data structure supports this algorithm. It contains user-provided pattern features and related values from which pattern class boundaries are defined. To produce this structure, one of several data representation algorithms is first applied to the user feature data. Class prototypes are defined. Then a heuristic feature subset selection algorithm is applied to these prototypes to reduce the size of the class defining data structure. This facilitates microprocessor implementation of the classifier. All of these algorithms were tested individually against various performance benchmarks before their implementation in the BOX80 system. Then as the system was developed, the algorithms were exercised as system modules were verified. Algorithms for data representation, classification and feature subset selection are discussed in this chapter. Related performance benchmarks, and testing procedures for system modules are presented as well.

Data Representation

To allow comparison of histogram displays between classes, and to enable byte sized component output for microprocessor use, scaling options are provided.

In creating the BOX80 feature vector data file, three scaling options are provided to standardize the range of component variation. These simplify later data comparisons. They are implemented by an energy, a unitizing, and a shifting transform. Each of these scaling options maintains relative angles between vectors. However, vector magnitudes vary. Given a feature vector F_n with components f_{nj}, these three options produce a new vector F_n' as follows.

Energy normalization:

$$F_n' = F_n/e \tag{4-1}$$

where $$e = \sum_{j=1}^{J} f_{nj}^2 \tag{4-2}$$

Unit normalization:

$$F_n' = F_n/|F_n| \tag{4-3}$$

where $$|F_n| = \left(\sum_{j=1}^{J} f_{nj}^2\right)^{1/2} \tag{4-4}$$

Shift normalization:

$$F_n' = mF_n + B \tag{4-5}$$

where $$m = 1/(a+b) \tag{4-6}$$

in which $a = \max \{ f_{nj} | n=1, N, j=1, J \}$

$b = -\min \{ f_{nj} | n=1, N, j=1, J \}$

and N = number of vectors in the data set

J = dimensionality of the feature space

and $$B = (b, b, \ldots b) \tag{4-7}$$

From the above, it is clear that each F'_n results from a linear shift of the original F_n. Therefore relative angles between the F'_n remain the same as the angles between the F_n. However, vector magnitudes do vary. For shift normalization there is a constant variation for the entire set $\{F_n\}$. For unit normalization, all vector magnitudes collapse to unity. In energy normalization while the energies of the F'_n become unity, their magnitudes become less than 1.

An additional transform is provided. This 'squaring' transform increases the precision possible in component values. However, it causes a twisting of the feature space which may change 'natural' relationships. It is provided as an input transform for experimentation only. This transform standardizes each feature component to the range apparent in the data set. This facilitates observation and measurement of data variation in each dimension of the feature space. Transformed vectors are produced as follows.

Squaring transform:

$$F'_n = F_n T^{-1} + B \qquad (4\text{-}8)$$

in which T = diagonal J x J matrix of t_{jj},

where $t_{jj} = (a_j + b_j)$ for

$$a_j = \max \{ f_{nj} | n=1, N \}$$

$$b_j = -\min \{ f_{nj} | n=1, N \}$$

and $B = (b_1, \ldots, b_J)$ for b_j as defined above.

In this transform both relative angles, and magnitudes of F_n' vary from those of F_n.

Normalization of feature component values using component variances measured from the user data set was considered as a possibility. Since there is some possibility that the distributions represented in that data set will not reflect those of the true population, this means of normalizing component values was not implemented. To cover the possibility that true population minimum and maximum values are not represented in the user data base, the ranges (a+b) referenced above can be extended by a fractional proportion with little problem.

In the generation of the microprocessor data structure which defines class boundaries, a transformation is necessary to map feature vector and prototype components into an eight bit range. Here, the squaring transformation of equation (4-8) is used since it preserves the greatest component precision. Since class boundaries exist at this point, no distortion of performance occurs. Use of this transformation implicitly assumes that it can be embedded into an independent feature generation process efficiently. This is a simple operation requiring only one add and one multiply for each feature.

In transforming class definitions there are two separate algorithms used. First, as given in equation (4-8),

$$F_n' = F_n T^{-1} + B.$$

Similarly, for class mean vectors, known as prototypes,

$$P_i' = P_i T^{-1} + B. \tag{4-9}$$

This prototype transform is readily derived at the vector level as follows:

$$P_i' = \frac{1}{L} \sum_{\ell=1}^{L} F_\ell' \tag{4-10}$$

$$= \frac{1}{L} \sum_{\ell=1}^{L} (F_\ell T^{-1} + B) \tag{4-11}$$

$$= (\frac{1}{L} \sum_{\ell=1}^{L} F_\ell) T^{-1} + B \tag{4-12}$$

$$= P_i T^{-1} + B \tag{4-13}$$

where

L = the order of class i

and T, B are defined as in (4-8)

The second algorithm operates on class boundaries. These are established by means of diagonal matrices referenced to the prototype vector. These matrices are explained in detail in the next section. To simplify this discussion of their transformation, consider class boundaries to have been defined by a diagonal class covariance matrix, Σ_i. The transformation for this class diagonal covariance matrix is clearly understood at the component level,

where j_{ij}' is the j^{th} component of Σ_i'

p_{ij}' is the j^{th} component of P_i'

$f'_{\ell j}$ is the j^{th} component of F'_{ℓ}

t_{jj} is the j^{th} member of T

b_j is the j^{th} member of B

$$\sigma'_{ij} = \{ \frac{1}{L} \sum_{\ell=1}^{L} (p'_{ij} - f'_{\ell j})^2 \}^{\frac{1}{2}} \quad (4\text{-}14)$$

$$= \{ \frac{1}{L} \sum_{\ell=1}^{L} \left(\frac{p'_{ij}+b_j}{t_{jj}} - \frac{f_{\ell j}+b_j}{t_{jj}} \right)^2 \}^{\frac{1}{2}} \quad (4\text{-}15)$$

$$= \frac{1}{t_{jj}} \{ \frac{1}{L} \sum_{\ell=1}^{L} (p_{ij}+b_j-f_{\ell j}-b_j)^2 \}^{\frac{1}{2}} \quad (4\text{-}16)$$

$$= \frac{1}{t_{jj}} \{ \frac{1}{L} \sum_{\ell=1}^{L} (p_{ij}-f_{\ell j})^2 \}^{\frac{1}{2}} \quad (4\text{-}17)$$

Thus

$$\sigma'_{ij} = \sigma_{ij}/t_{jj} \quad (4\text{-}18)$$

This transformation is provided as an option prior to the calculation of classification error rates. The option, through its use of integer calculations, allows simulation of microprocessor performance by the BOX80 system. The transformation is also exercised prior to output of the class defining data structure in microprocessor format. This allows byte sized encoding of output component values.

Classification Algorithm

A feature vector associated with an unknown pattern is assigned to a known data class by a classification algorithm. The BOX80 system classification algorithm partitions hyperspace

into regions which can be visualized as hyperspace boxes. Class membership is derived from the identifier of the hyperspace box which contains the unknown feature vector. Since these boxes need not necessarily be mutually exclusive of one another, the containment property is obtained through a distance measurement with which decision ambiguities are resolved.

The BOX80 classification algorithm was designed to maximize operating efficiency within a microprocessor implementation. Minimum use of memory, as required, reduces execution time. This algorithm was also designed with the number of feature dimensions and the number of pattern classes as parameters of its execution. Any combination of I classes and J feature dimensions can be processed given that sufficient memory is available.

The algorithm is implemented within both of the BOX80 system segments. There are small variations between these implementations. In one instance the implementation is in FORTRAN. Here, the referenced data structure is a two-dimensional array containing a collection of vectors. Each class is defined by a set of three of these vectors. Two options are provided this implementation. One uses a Euclidean norm for the distance measurement rather than the supremum norm. The other option enables processing of scaled data. It substitutes truncated integers for real values of referenced vector components. In the second instance the algorithm has no options. Its referenced

data structure is a linear list partitioned into a series of segments, one for each data class. This instance occurs in the micro-processor based classifier routine. It is written in the assembly language for the 8080. (Ref 16)

Memory requirements for data used by the above two implementations of the BOX80 classifier are calculated in terms of the numbers of classes (I) and features (J) to be processed. Memory (M) required for the Interpreter Segment's FORTRAN data structure is

$$M = (J+3)\ (2I+K) \tag{4-19}$$

Memory required for the 8080 Classifier Segment implementation is calculated

$$M = [(3J)+1](I) \tag{4-20}$$

The FORTRAN implementation references a data structure in which vector dimensionality has been increased by three extra values. This produces the factor (J+3). The factor K indicates the number of classes having asymmetric boundaries. This differs with the 8080 implementation which adds only one extra value, a class identifier, to each class. This implementation assumes that each class has asymmetric boundaries.

The algorithm implements a variation of the minimum distance classification rule. An unknown vector is assigned membership in that class to which it is nearest. However, this algorithm exhibits facets of other common classifier algorithms. From the perspective that the algorithm references the multivariate

covariance of each class' features, it can be considered a variant of a Bayesian decision rule. However, no formulation of the a priori probability of class membership is made. Furthermore, feature dimensions must be assumed to present uncorrelated, independent measurements of pattern variation. Finally, these feature measurements must be assumed to be completely representative of pattern class membership and must be assumed to generate Gaussian distributions. Therefore, although the algorithm has a statistical flavor, it is not a true Bayesian algorithm. However, from the standpoint that its referenced data structure partitions the feature space into a collection of hyperspace boxes each of which bounds a neighborhood of a given class, it can be considered a variant of a condensed nearest neighbor rule. This perspective is justified by the fact that each class boundary is statistically constructed so as to enclose an advantageous subset of class members. Here, in discriminating between classes to produce the classification assignment, the evaluation of distances to class boundaries is analogous to evaluation of distances to the nearest neighbors of the unknown pattern. The weakness in this comparison lies in the fact that the BOX80 algorithm tends to benefit from convex class boundaries. The NN algorithm needs no such assumption.

The data structure which establishes each class' boundaries consists of a vector and a pair of diagonal matrices. The vector

is a class mean or prototype vector. For class i consisting of a set ω_i of feature vectors F_ℓ of dimensionality J, this prototype vector is

$$P_i = \frac{1}{L} \sum_{\ell=1}^{L} F_\ell. \tag{4-21}$$

The two diagonal matrices establish class boundaries in terms of component variation from this mean. These matrices are most clearly defined at the component level. Consider a class of feature vectors represented by L members of dimensionality J. A feature vector within ω_i is

$$F_\ell = (F_{\ell 1}) \ldots f_{\ell j}, \ldots f_{\ell J}) \tag{4-22}$$

and the prototype vector for the class is

$$P_i = (P_{i1}, \ldots P_{ij}, \ldots P_{iJ}) \tag{4-23}$$

The diagonal matrix which establishes boundaries less than this prototype is

$$Z^{-i} = \begin{bmatrix} \ddots & & 0 \\ & z_{jj} & \\ 0 & & \ddots \end{bmatrix}_{J \times J} \tag{4-24}$$

The diagonal matrix which establishes boundaries greater than the prototype is similarly represented

$$Z^{+i} = [z^{+}_{jj}]. \tag{4-25}$$

Note that the subscripts of matrix components do not reflect membership in class i. This is simply a convenient notation.

These components are formed as follows.

$$\text{iff } f_{\ell j} > P_{ij},\ z^{+}_{jj} = \left[\frac{1}{L} \sum_{\ell=1}^{L} (p_{ij}-f_{\ell j})^2 \right]^{\frac{1}{2}} \tag{4-26}$$

$$\text{iff } f_{\ell j} \leq P_{ij},\ z^{-}_{jj} = \left[\frac{1}{L} \sum_{\ell=1}^{L} (p_{ij}-f_{\ell j})^2 \right]^{\frac{1}{2}} \tag{4-27}$$

In defining a class in terms of a class mean vector and two boundary matrices, a minimum Euclidean distance algorithm can be constructed. However, a scaled distance measurement is used here. That is, the distance of an unknown vector from a class prototype will be measured in each component dimension in terms of a number of boundary units. This is a distance measure similar to the Mahalanobis distance. Given uncorrelated features, and using the simplying assumption made for equations (4-14) to (4-18)

$$P_r(F_n \in \omega_i) \geq \prod_{j=1}^{J} (1-\sigma_j^2). \tag{4-28}$$

Where the features are correlated, this probability can be written

$$P_r(F_n \in \omega_i) \geq \max_{j} \left(0, \left(1- \sum_{j=1}^{J} \sigma_j^2\right)\right) \tag{4-29}$$

These bounds are derived from Tchebychef's inequality by Godwin (Ref 12:63).

To assign class membership to an arbitrary feature vector F_n with components f_j, first a composite boundary matrix, Z^i, is formed for each class i. This produces

$$Z^i = [z_{jj}^{(i)}]. \tag{4-30}$$

In this composite boundary matrix

$$\text{iff } f_j > p_{ij} \text{ then } z_{jj}^{(i)} = z_{jj}^{+} \tag{4-31}$$

and $$\text{iff } f_j \leq p_{ij} \text{ then } z_{jj}^{(i)} = z_{jj}^{-}. \tag{4-32}$$

Distance from an unknown F_n to this class is next computed, first as a vector and then as a scalar. This effects a classifying decision rule as follows

$$D_{in} = (P_i - F_n)\, Z^i \tag{4-33}$$

$$d_{in} = ||D_{in}||. \tag{4-34}$$

The scalar d_{in} is considered a member of the set

$$\Delta^* = \{d_{in}, \ldots d_{1n}, \ldots d_{In}\}. \tag{4-35}$$

Class membership is then assigned to that class to which distance is minimum. That is

$$d_k = \underset{i}{\text{MIN}} \{d_{in}\} \rightarrow F_n \in \omega_k. \tag{4-36}$$

Several notes about this algorithm are worthwhile. The two-sided approach to defining class boundaries was suggested by Pacheco (Ref 32:11) in the course of a review of the radar signature recognizer described in Chapter 2. The simpler process which uses a single boundary matrix to define both sides of a symmetric hyperspace boundary for a class can be described as a minimum distance classifier having a Mahalanobis' distance metric. The assumption that feature dimensions are uncorrelated and

therefore independent allows the composite boundary matrix, Z^i, to be considered as a diagonal covariance matrix, Σ_i. In this case the distance measurement to the i^{th} class can be written.

$$d_{in}^2 = (F_n - P_i)\, \Sigma_i^{-1}\, (F_n - P_i)^T. \tag{4-37}$$

The equivalence of this expression to equation (4-33) is readily seen in a simple example. Let dimensionality J=2, and

$$X = P_i - F_n \tag{4-38}$$

where $X = (p_{i1}-f_{n1},\ p_{i2}-f_{n2})$. (4-39)

$$\text{Let } \Sigma^{-1} = \begin{bmatrix} 1/\sigma_{11}^2 & 0 \\ 0 & 1/\sigma_{22}^2 \end{bmatrix}_{J \times J = 2 \times 2} \tag{4-40}$$

$$\text{where } \sigma_{jj}^2 = \left(\frac{1}{L} \sum_{\ell=1}^{L} (p_{ij}-f_{\ell j})^2\right) \tag{4-41}$$

In this example it is notationally clear that

$$d_{in}^2 = [x_1,\ x_2] \begin{bmatrix} 1/\sigma_{11}^2 & 0 \\ 0 & 1/\sigma_{22}^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{4-42}$$

From the rules of matrix algebra, this is

$$d_{in}^2 = [x_1/\sigma_{11}^2,\ x_2/\sigma_{22}^2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{4-43}$$

which is

$$d_{in}^2 = x_1^2/\sigma_{11}^2 + x_2^2/\sigma_{22}^2. \tag{4-44}$$

Equation (4-44) defines the square of the Eucledian norm in two space. Thus one sees that

$$d_{in}^2 = |D_{in}|^2 \tag{4-45}$$

where $D_{in} = [x_1, x_2] \begin{bmatrix} 1/\sigma_{11} & 0 \\ 0 & 1/\sigma_{22} \end{bmatrix}$

which is

$$D_{in} = (P_i - F_n)\, Z^i. \tag{4-47}$$

In this way the equivalence of equations (4-37) and (4-33) has been demonstrated.

The foregoing presentation of the BOX80 classification algorithm avoids one issue and glosses over another. The former is a programmatic statement of the actual algorithm which references the defined computer data structure. This is presented at the close of this chapter. The latter is the derivation of the BOX80 nomenclature. This explanation follows.

(1) The J dimensional region defined by equation (4-37) forms an ellipsoid in hyperspace whose shape is specified by Σ_i (Ref 13:36). This ellipsoid has its axes oriented along the axes of the space since Σ_i is diagonal.

(2) The J dimensional region defined by equation (4-33) forms a hyperrectangle about the prototype vector, P_i. This results from a computationally simplifying norm used to produce the magnitude of D_{in}. This norm is defined as follows

$$||D_{in}|| = \sup (x_i, \ldots x_j, \ldots x_J) \tag{4-48}$$

where $D_{in} = (P_i - F_n) = (x_i, \ldots x_j, \ldots x_J)$ (4-49)

This norm produces a well-defined metric and is well known for its computational simplicity (Ref 7:104). It can be shown that in the limit

$$\lim_{p\to\infty} \left[\sum_{j}^{J} |x_j|^P \right]^{1/P} = \max |x_j| \tag{4-50}$$

(3) The region bounded by the vector pair

$$U^R = P_i + P_i Z^{+i} \tag{4-51}$$

and $$U^L = P_i + P_i Z^{-i} \tag{4-52}$$

encloses a subset of $F_\ell \in \omega_i$. Fig 4 describes this region for $\omega_{i=1}$ and $\omega_{i=2}$ in a space having J=2 dimensions.

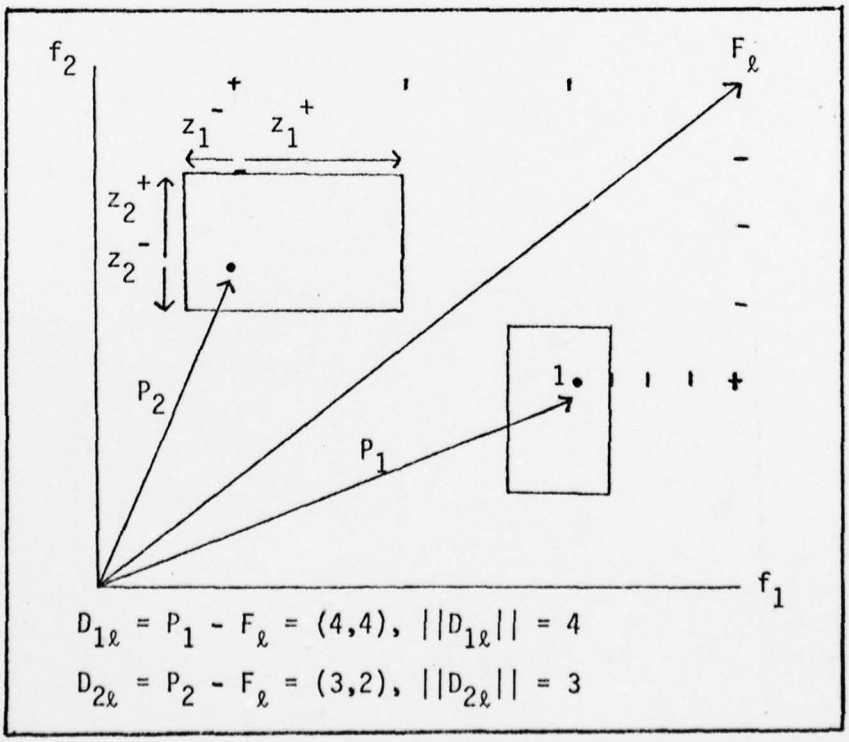


Fig. 4. BOX80 Distance Measure

The rectangular aspect of these class regions, from the sup norm distance metric, becomes clear in this figure.

Classification Algorithm:

```
 1. procedure CLASS (FEAT(I,L),IC)
 2. begin
 3.   set DMIN = 1E10
 4.   for all classes I do
 5.     begin
 6.       set DMAX: = -1E10
 7.       set NCAV (to index class I, P_i)
 8.       set NCSDL (to index class I, Z^-i)
 9.       set NCSDR (to index class I, Z^+i)
10.       if NCSDR eq 0 then
11.          set NCSDR: = NCSDL
12.       for all dimensions J do
13.          begin
14.             set NCSD:NCSDL
15.             set DFEAT:=CLAS(J,NCAV)-FEAT(J,L)
16.             if DFEAT gt 0.0 then
17.                set NCSD:=NCSDR
18.             set DFEAT:= DFEAT/CLAS(J,NCSD)
19.             if ABS(DFEAT) gt DMAX then
20.                set DMAX:=ABS(DFEAT)
21.          end 'J'
```

```
22.          if DMAX lt DMIN then
23.             set IC:=I
24.       end "I"
25.    "BOX80 CLASSIFIER"
26.  end "CLASS"
```

Feature Selection

Good features make good pattern recognizers. Unnecessary pattern features make inefficient pattern recognizers. Thus, identifying the best features is important to developing an acceptable pattern recognizer.

The literature reflects considerable work done to solve the general problem of identifying features. This problem may be approached in one of three ways. Firstly, one may rely upon analytical theory to identify just the set of features which should be extracted from the pattern environment. However, theory does not always identify a set of measurements which suffice to completely classify a pattern environment. In another technique, one may compute a large set of candidate features and then rely on transforms and filters prior to classification to generate a smaller set of significant factors. In a third method, one may evaluate a candidate set of features in the light of a classification algorithm, and preselect the most desirable subset. The recognizer then operates directly on this subset of features without added processing.

An assumption underlying this thesis has been that convenience and efficiency are more critical factors in developing an initial recognition model than a proven optimality or a comprehensive analytical basis. Stearns (Ref 34:71) notes that from the standpoint of hardware, reducing the original set of measurements by principal component analysis and transforms may even produce a loss in overall system performance. His argument allows that when a transform to a subspace is effected, all features of the original space have to have been generated. Thus, even though subsequent processing may benefit by reduced subspace dimensions, the computational costs of feature extraction must still be carried. This argument led to the development of a subset selection algorithm to implement the BOX80 system.

Prior selection of an acceptable subset of features has advantages for microprocessor implementations of distributed pattern recognition systems. In such a system the master processor can be used to extract features from the environment. Its feature extraction software may initially be coded to generate many feasible and reasonable pattern characteristics. The slave processor can be used to execute a pattern classifier and produce recognition decisions. Once a subset of features has been selected by a process such as that supported by the BOX80 Interpreter Segment, the feature extraction algorithm can be streamlined by straightforward deletion of extraneous computations.

The result is a process which uses less time. Then the new class defining data structure is provided to the classifier and another data-gathering, recognizer evaluation cycle can begin.

Search algorithms for finding a better subset of features have two common elements as described in chapter II. Estimation of error probability by calculation of an empirical error rate is the best evaluation for any feature subset. Before a subset can be evaluated, it must be constructed by a mapping from the original feature set. The BOX80 system does not implement a search algorithm. Instead, the search iteration is opened to the user. Thus, the user can specify the mapping which creates the subset to be tested. He can also control the search iteration by his evaluation of the empirical error rate which applies to the subset of interest.

In order to guide the user towards selection of trial subsets of features, a figure of merit is calculated for each feature. This figure of merit reflects the contribution that its associated feature makes to the recognition decision. To establish this contribution a set of interclass distance vectors are computed. Combinations of these vectors produce diagonal matrices whose components are the figures of merit for their respective feature dimensions. Three different matrices are computed based upon the distance measurement of equation (4-33). In this case, a prototype vector representing an 'unknown' class is substituted

for the unknown feature vector of the original equation. Two sets of matrices $\{D_{in}\}$ and $\{D_{ni}\}$ are computed for each class as follows.

$$X_{in} = (P_i - P_n)\, Z^i \tag{4-53}$$

and $$X_{ni} = (P_n - P_i)\, Z^n \tag{4-54}$$

where $1 \leq i \leq I$,

$1 \leq n \leq I$

and matrices Z^i and Z^n are established as for equation (4-33). A diagonal matrix is constructed from each of the vectors X_{in} and X_{ni} simply by considering the vector components as the appropriate members of the matrix' diagonal. Thus

$$D_{in} \leftarrow X_{in}$$

and $$D_{ni} \leftarrow X_{ni} \ .$$

The set of matrices $\{D_{in} | 1 \leq n \leq I\}$ establish the distances <u>from</u> class i <u>to</u> each of the other class prototypes in the data structure. Components of these diagonal matrices are measured in the boundary units of class I. On the other hand, the set matrices $\{D_{ni} | 1 \leq i \leq I\}$ reflect the opposing distances <u>to</u> class i <u>from</u> each of the other class prototypes in the data structure. Components of these diagonal matrices are measured in the boundary units of each of the "other" classes. Opposing matrices D_{in} and D_{ni} are rarely the same which indicates that this distance measurement does not form a metric on the discrete space of prototype vectors.

A series of experiments was used to evaluate these interclass

distances. One set of merit figures resulted from each experiment. A measure of the 'volume' of each class was sought. Three were produced. For the first (subscripted 3 below to match program code), a distance matrix was calculated for each class,

$$V_i = \sum_{n=1}^{I} (D_{in} + D_{ni}),\ n \neq i. \tag{4-55}$$

Then a merit matrix for the feature space was derived from these V_i:

$$M_3 = \sum_{i=1}^{I} V_i \ . \tag{4-56}$$

The components of this diagonal matrix became figures of merit for their respective feature dimensions.

Each component of the diagonal matrix, M_3, is related to the total interclass distance in its dimension. Experimentation with these component values as merit figures led to the realization that overlap between a pair of classes in a given dimension was not as well reflected in this figure of merit as possible. This can be seen in a numerical example. Let

$$V_1 = \begin{bmatrix} 16.0 & & 0 \\ & 16.0 & \\ 0 & & 8.0 \end{bmatrix} \quad \text{and} \quad V_2 = \begin{bmatrix} 1.0 & & 0 \\ & 0.5 & \\ 0 & & 9.0 \end{bmatrix},$$

for a 2 class, 3 dimension instance. Note that in this case

$$V_i = D_{in}.$$

Here the merit matrix is

$$M_1 = \begin{bmatrix} 17.0 & & 0 \\ & 16.5 & \\ 0 & & 17.0 \end{bmatrix}$$

and the differences among the feature merits, m_{jj}, are not appreciable. However, the components of the postulated V_i show that in feature dimension 3 the classes are almost equally separated at large distances of 8.0 and 9.0 boundary units. Therefore, the classes are readily separable in this dimension. This is clear from the operation of the classifier algorithm which computes for this dimension,

$$\Delta = \{d_{13}, \ldots d_{i3}, \ldots d_{I3}\} \tag{4-57}$$

in which

$$d_{i3} = (p_{13} - p_{23})/z_{33}^{(i)}. \tag{4-58}$$

Allowing that $z_{33}^{(i)} \sim \sigma_{33}^{(i)}$ for symmetric classes, and considering each class in turn as the unknown,

$$(p_{13} - p_{23}) = 8.0\ \sigma_{33}^{(1)}$$

and $$(p_{23} - p_{13}) = 9.0\ \sigma_{33}^{(2)}.$$

In a Tchebyshev sense there is little likelihood of confusion between the two classes in dimension 3. However, similar computations for dimension 1 indicate

$$(p_{11} - p_{21}) = 16.0\ \sigma_{11}^{(1)}$$

and $$(P_{21} - p_{11}) = 1\ \sigma_{11}^{(2)}.$$

Here, in the same sense, the likelihood of confusion between classes is great. A similar condition exists to an even greater degree in dimension 2. To rectify this situation another set of merit figures was computed as follows.

$$M_2 = \prod_{L=1}^{I} V_L \qquad (4\text{-}59)$$

In this case, the components of the diagonal matrix M_2 are more sensitive to the appearance of a small component within some matrix V_i. Using the V_i matrices of the previous example this M_2 matrix is

$$M_2 = \begin{bmatrix} 16.0 & 0.0 & 0.0 \\ 0.0 & 8.0 & 0.0 \\ 0.0 & 0.0 & 72.0 \end{bmatrix} .$$

Here there is clear indication of the strength of feature 3. The appearance of the relatively small values 8.0 and 16.0 at components m_{11}, $m_{22} \in M_2$ indicate that in these features many classes are relatively close to one another. However a given feature may discriminate well between all but one class. This instance is not reflected well by the components of M_2. Thus a third merit matrix was generated. This is

$$M_1 = \sum_{i=1}^{I} \ln \left(\prod_{m=1}^{I} D_{in} \right), \; n \neq i. \qquad (4\text{-}60)$$

This formulation differs from the earlier ones in the use of a logarithmic sum, and in the use of matrices D_{in}, only. Explanation follows.

(1) The logarithmic sum produces merit figures which form the same ordered sequence by magnitude as the merit figures produced by the matrix product.

$$M_1' = \prod_{i=1}^{I} \left(\prod_{n=1}^{I} D_{in} \right), \; n \neq i \qquad (4\text{-}61)$$

However the values of the logarithmic sum are not nearly so likely to overflow the floating point limit of the computer. It was hypothesized that this matrix product formulation would reflect dimensions having single class confusion by a greater variation in its components than there would exist among components of M_2. (The notion was that in the double product, components would change geometrically, while in the sum of products they would vary arithmetically). Testing with merit figures from M_2 to M_1 is reported in the next section. Some experimenting, in a three class problem, was done with the M_1' figures of merit. These appeared more robust than M_2 figures. However, the 26-class alphabet problem created overflow in the M_1' matrix. The M_1 merit figures, as can be seen in the next section, do not reflect the robustness of the M_1' figures.

(2) Merit matrix M_1 is formed from matrices D_{in} only, since this produces results equivalent to those obtained with the matrix sum $(D_{in} + D_{ni})$ as for matrix M_{21}. This is because

$$\prod_{i=1}^{I} (D_{in}) = \prod_{n=1}^{I} (D_{ni}), \; i=n \qquad (4\text{-}62)$$

These procedures for establishing merit figures for feature dimensions have a similar basis to those of Michael and Lin (Ref 28:172). They produce a means of ordering features in terms of capacity to discriminate between classes. They are intended only as a starting

point for a heuristic, manually controlled search for a good subset of features.

To establish subsets of features, the BOX80 system uses a mapping algorithm which maps the original feature space into a subspace. This mapping process references an ordered list of feature dimension tags. Each tag is the number of a feature in the original space. An ordering may be constructed by sorting these feature tags by their respective merit figures. An arbitrary order may also be manually input. The mapping algorithm is imbedded in a routine which computes error rates for J different subspaces. These error rates can be generated during a single iteration of the trial classifier. The process of constructing tentative feature subspaces is thus piggybacked onto the BOX80 performance evaluation function.

Subspaces constructed by the mapping algorithm are based on a nesting of proper subsets of features. These subsets contain an increasing number of features from 1 to J. Each subset is contained by its successor.

In the classification procedure described earlier, a distance vector is calculated. This is

$$D_{in} = (P_i - F_n)\, Z^i \qquad (4\text{-}63)$$

The mapping algorithm operates on the components of this vector to produce a set of J nested subsets, S_j. An error rate is computed for each of these. An example may clarify the process. Let J=3 and (3,1,2) be a list of feature tags ordered by figure

of merit. Let the distance vector

$$D_{in} = (14.0, 63.1, 9.0).$$

Here, the mapping algorithm constructs

$$S_1 \subset S_2 \subset S_3$$
$$(9.0) \quad (9.0, 14.0) \quad (9.0, 14.0, 63.1)$$

as the set of nested subsets. Each of these is considered a distance vector in its respective subspace of the original three-dimensional space. The decision rule is operated on each of these vectors at once. This is the key point. Rather than operate the decision rule on each vector in series, these nested vectors are processed in parallel. Since the max and min functions which implement the decision rule can be done in a parallel fashion, some execution cost is saved. Thus, for each j, $1 \leq j \leq J$,

$$d_{jk} = \min_{1} \{||S_j||, 1 \leq i \leq I\} \rightarrow S_j \in \omega_k,$$

and a class assignment is obtained and an error rate is computed for each subspace.

Finally, a special procedure, termed a zapping process, is used to modify the tentative class definition structure to establish a chosen subspace as the basis for future trial recognition experiments. In this process, selected components of all members of the set of Z_i^+ and Z_i^- matrices (which reflect class boundaries) are increased to large values in each matrix. The effect is to nullify all measurements made in those dimensions.

The algorithm used for computation of merit figures, and the algorithm used to map and evaluate feature subspaces are

presented in the following two paragraphs. The former is titled MERIT. The latter is termed LOOK.

Algorithm for Mapping and Subspace Evaluation:

```
1.  Procedure LOOK[DIS(J),ITAG(J),RATE(J),NEW,KNOW,I]
2.  begin
3.     if NEW eq 1 then
4.     begin
5.         for all J do
6.         begin
7.           set CLOSE(J) = 1E9
8.           set ISAV(J) = 0
9.         end
10.    end
11.    for all J do
12.    begin
13.        set K = ITAG(J)
14.        set WORK(J) = DIS(K)
15.    end "J"
16.    for all J do
17.    begin
18.        set RMAG = -1E20
19.        for K from 1 to J do
20.        begin
21.          if WORK(K) ge RMAG then
```

```
22.                 set RMAG = WORK(K)
23.             end "K"
24.             if RMAG le CLOSE(J) then
25.               begin
26.                 set IPICK(J) = I
27.                 set CLOSE(J) = RMAG
28.               end
29.             if NEW eq 2 then
30.               if IPICK(J) eq KNOW then
31.                 set RATE(J) = RATE(J) + 1.
32.             end
33.         end
34.     end
```

Algorithm for Figures of Merit:

```
1.  procedure MERIT [CLAS(J,I),FT(J,5)]
2.  begin
3.      for all J do
4.      begin
5.          set FT(J,1): = FT(J,2): = FT(J,4): = FT(J,5): = 1.0
6.          set FT(J,3): = 0.0
7.      end "J"
8.      for all I do
9.      begin
```

```
10.         set ICAV (to index CLASS I, P_i)
11.         set ICSDL (to index CLASS I, Z_i^-1)
12.         set ICSDR (to index CLASS I, Z_i^+1)
13.         if ICSDR eq 0 then
14.             set ICSDR: = ICSDL
15.         for all N except N=I do
16.         begin
17.             set NCAV (to index CLASS N, P_n)
18.             set NCSDL (to index CLASS N, Z_n^-n)
19.             set NCSDR (to index CLASS N, Z_n^+n)
20.             if NCSDR eq 0 then
21.                 set NCSDR: = NCSDL
22.             for all J do
23.             begin
24.                 if J eq 1 then begin
25.                     set FT(J,4) = 1.0
26.                     set FT(J,5) = 0.0 end
27.                 set DI(J): CLAS(J,ICAV)-CLAS(J,NCAV)
28.                 set DN(J) = DI(J)
29.                 set ICSD: = ICSDL
30.                 set NCSD = NCSDL
31.                 if DI(J) lt 0 then begin
32.                     set ICSD: = ICSDR else
33.                     set NCSD: = NCSDR end
```

```
34.                 set DI(J):= DI(J)/CLAS(J,ICSD)
35.                 set DN(J):= DN(J)/CLAS(J,NCSD)
36.                 set FT(J,3):= FT(J,3)+DI(J)+DN(J)
37.                 set FT(J,4):= FT(J,4)*DI(J)
38.                 set FT(J,5):= FT(J,5)+DI(J)+DN(J)
39.             end "J"
40.         end "N"
41.         for all J do
42.         begin
43.             set FT(J,1):=FT(J,1)+Ln (FT(J,4))
44.             set FT(J,2):=FT(J,2)*FT(J,5)
45.         end "J"
46.     end "I"
47. end "Merit"
 *.     FT(J,3) contains figures of merit M3
 *.     FT(J,4) contains figures of merit M1
 *.     FT(J,5) contains figures of merit M2
```

$$M_1 = \sum_{i=1}^{I} \ln \left(\prod_{n=1}^{I} D_{in} \right), \; n \neq i$$

$$M_2 = \prod_{i=1}^{I} \left[\left(\sum_{n=1}^{I} (D_{in} + D_{ni}) \right) \right], \; n \neq i$$

$$M_3 = \sum_{i=1}^{I} \left[\left(\sum_{n=1}^{I} (D_{in} + D_{ni}) \right) \right], \; n \neq i$$

Performance Benchmarks

The BOX80 system is a designer's tool. It is intended for student use in development of experimental pattern recognition systems. It produces a class-defining data structure upon which a microprocessor based pattern classifier can operate. BOX80 system performance is relfected in the error rate of its classifier. This error rate is heavily dependent upon the nature of the data set from which the class defining data structure is derived. However, the BOX80 system's algorithms and procedures do contribute to this performance. No argument is made here that these algorithms are optimum. Nor is it claimed that BOX80 system procedures are uniquely effective. Nevertheless, these algorithms and procedures are sufficient to generate class defining data structures efficiently and effectively. These claims are supported by the discussion following.

System Efficiency. Here, the cost-benefit trade-off is critical. It makes no sense to me to optimize a classifier algorithm on the basis of a data set, however extensive, which cannot be proven optimal. In the recognition of electromagnetic patterns, sample data collection is biased almost by definition. Sensor locations may be constrained; hardware transients may be unpredictable; the pattern environment may even be simulated. The BOX80 system is configured to provide a low cost avenue towards the necessary class defining data structure. Finally, the BOX80

classifier itself is configured for low-cost microprocessor implementation.

(1) In generating a class defining data structure, the BOX80 system uses a system segment of four programs. These programs optimize memory use with generalized data structures and a memory allocation module. They communicate through standard system data files. These files and program source code conform to ANSI standards. Program structure is modular. Design conforms to top-down concepts. As a result, this system segment is transportable, and readily modifiable. Since it can be readily configured for use on any minicomputer or large-scale system, it is a low cost tool for use in pattern recognizer development. The efficiency of the individual programs in this segment is not as critical as the above general cost of using the system. Yet, in the alphabet classification experiment discussed in this section, the trial classification process required less than 55K of CDC6600 memory and executed in less than 23 cpu seconds. This contrasts to the similar costs of 140K memory and 41 cpu seconds for the specialized alphabet classifier program which provided comparison data.

(2) The classifier segment of the system uses less than 256 bytes of microprocessor ROM. The class defining data structure, of course, uses RAM memory in relation to its size as specified in equation (4-20). No actual timing of the execution of this segment has been performed. To some extent this timing is problem-

dependent. That is, the total time required to iterate through the data structure for a given problem depends on the numbers of classes and features for that problem. In addition, the very simplicity of this algorithm indicates a speedy execution.

System Effectiveness. Here, the contribution to performance of system algorithms and procedures is addressed. The classifier algorithm operates with an error rate within reasonable limits of that produced by a comparable algorithm on each of two data sets. Similarly, the algorithm which evaluates feature merit establishes merit figures which match, within limits, the merit figures established by other such algorithms on these data sets. Finally, the procedures for selection of a feature subset, and for generation of the class defining data structure for a microprocessor, successfully reduce data structure size without increasing the classifier error rate significantly. These aspects of system performance are detailed in the following paragraphs.

Previous thesis work at AFIT produced the two data sets with which BOX80 system performance has been evaluated (Refs 33, 24). Performance benchmarks were established for each data set. BOX80 system algorithms were analyzed in terms of these benchmarks both during design and after implementation. This analysis follows.

(1) Table I and Figs 5 to 16 apply to Frequency of Occurrence of Binary Words data. This data consists of some 500 feature vectors of 14 components which represent patterns from a

TABLE I.
PEARSON CORRELATION COEFFICIENTS
FOBU DATA SAMPLE-1

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14
F1	1.0000	.8951	-.8590	.8610	-.6813	-.8589	-.8255	.8029	-.7264	.6627	-.8303	-.7224	-.6590	-.6172
F2	.8951	1.0000	-.7377	.8081	-.7177	-.9240	-.7569	.7842	-.7137	.7156	-.8104	-.7519	-.5824	-.6037
F3	-.8590	-.7377	1.0000	-.9308	.6810	.7115	.8921	-.8616	.7577	-.6631	.8917	.7519	.7860	.6844
F4	.8610	.8081	-.9308	1.0000	-.8102	-.7779	-.9552	.9329	-.8696	.7899	-.9519	-.8599	-.8389	-.7921
F5	-.6813	-.7177	.6810	-.8102	1.0000	.6918	.7774	-.8728	.9450	-.9728	.7710	.9331	.7691	.8903
F6	-.8589	-.9240	.7115	-.7779	.6918	1.0000	.8343	-.6780	.7691	-.6039	.6668	.6251	.4919	.5252
F7	-.8255	-.7569	.8921	-.9552	.7774	.8343	1.0000	-.8358	.8962	-.6920	.8256	.7491	.7502	.7154
F8	.8029	.7842	-.8616	.9329	-.8728	-.6780	-.8358	1.0000	-.8697	.8966	-.9514	-.9559	-.8980	-.8744
F9	-.7264	-.7137	.7577	-.8696	.9450	.7691	.8962	-.8697	1.0000	-.8728	.7647	.8863	.7989	.8533
F10	.6627	.7156	-.6631	.7899	-.9728	-.6039	-.6920	.8966	-.8728	1.0000	-.8188	-.9689	-.8110	-.9020
F11	-.8303	-.8104	.8917	-.9519	.7710	.6668	.8256	-.9514	.7647	-.8188	1.0000	.8981	.8513	.7961
F12	-.7224	-.7519	.7519	-.8599	.9331	.6251	.7491	-.9559	.8863	-.9689	.8981	1.0000	.8758	.9132
F13	-.6590	-.5824	.7860	-.8389	.7891	.4919	.7502	-.8980	.7989	-.8110	.8513	.8758	1.0000	.9264
F14	-.6172	-.6037	.6844	-.7921	.8903	.5252	.7154	-.8744	.8533	-.9020	.7961	.9132	.9264	1.0000

three-class recognition problem. Both the Online Pattern Analysis and Recognition System (OLPARS) (Ref 5) and the Statistical Package for the Social Sciences (Ref 30) were used to establish error rates for the classification of this data.

(a) As in other radar pattern recognition problems, the features in this data set are highly correlated. Table I presents Pearson Correlation coefficients. These represent an index of the degree of linear relationship between the features. (Ref 27). As can be seen, fewer than twenty percent of the meaningful correlations are less than .70. Note that only one of these is less than .50 and that nearly half of these associate with feature 6.

(b) Using a Mahalanobis' distance based discriminant analysis procedure (DISCRIMINANT), SPSS produced an overall classifier error rate of 26.6 percent. (See Fig 5.) The OLPARS system also processed this data. With the same statistical measure, its nearest mean vector procedure (NMV) produced an error rate of 27.7 percent. (See Fig 6.) The BOX80 system error rate, 34.5 percent, is shown in Fig 7. To interpret this figure, notice that the summary conclusion values are a percent correctly classified, a percent classified in error, and a percent rejected. Rows of the BOX80 confusion matrix contain a count of data vectors belonging to the class, the class id, and standard confusion matrix assignment percentages. Other data output is discussed in chapter 5. Note that SPSS and OLPARS algorithms use a process

CENTROIDS OF GROUPS IN REDUCED SPACE

GROUP 1	-1.40581	-.00974
GROUP 2	1.09814	-.52665
GROUP 3	.99107	.23744

DISCRIM FOBW DATA

PREDICTION RESULTS -

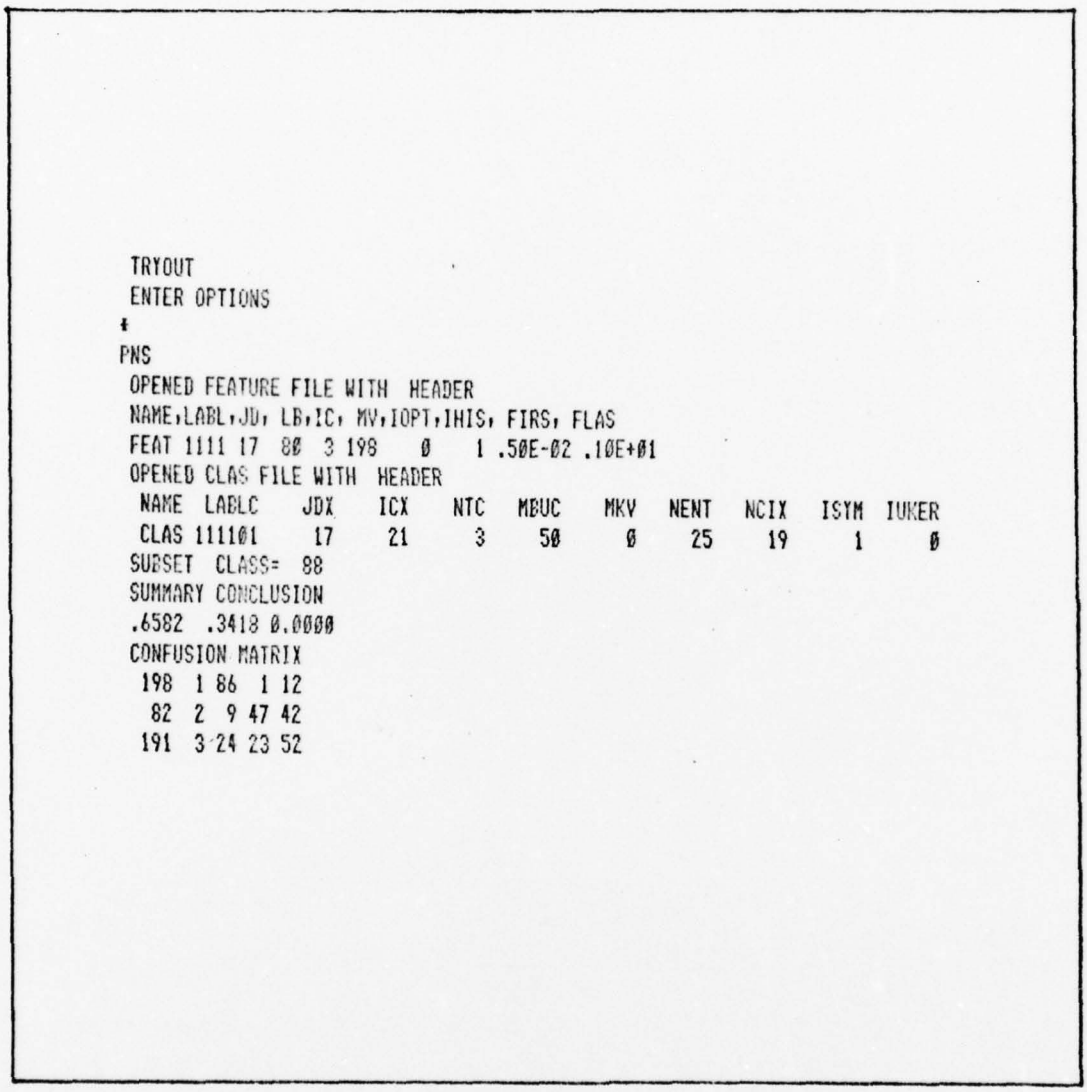
ACTUAL GROUP NAME	CODE	N OF CASES	PREDICTED GROUP MEMBERSHIP GROUP 1	GROUP 2	GROUP 3
GROUP 1	1	198	166.	10.	22.
			83.8 PCT	5.1 PCT	11.1 PCT
GROUP 2	2	82	2.	54.	26.
			2.4 PCT	65.9 PCT	31.7 PCT
GROUP 3	3	190	10.	55.	125.
			5.3 PCT	28.9 PCT	65.8 PCT

73.4 PERCENT OF KNOWN CASES CORRECTLY CLASSIFIED

FIG 5. SPSS DISCRIMINANT RESULTS

```
Overall Evaluation:
Dataset kdigrams **** passed against logic designed on firshalf
Number of dimensions = 14


      true class

      AAAA BBBB CCCC
AAAA   190   24   51
BBBB     1   12    5
CCCC     8   46  151
rejt     0    0    0

totl   199   82  207
corr   190   12  151
%cor  95.5 14.6 73.0
eror     9   70   56
%err   4.5 85.4 27.1
rejt     0    0    0
%rej   0.0  0.0  0.0


total number of vectors = 488
overall correct    353 for  72.34%
overall error      135 for  27.66%
overall reject       0 for   0.00%



Overall Evaluation Summary:
Dataset kdigrams **** passed against logic designed on firshalf
Number of dimensions = 14

node    %c     %e     %r

AAAA 95.48  4.52  0.00   overall correct
BBBB 14.63 85.37  0.00        72.34%
CCCC 72.95 27.05  0.00   overall error
                              27.66%
                         overall reject
                              0.00%
```

FIG 6. OLPARS NMV ERROR RATE



```
 TRYOUT
 ENTER OPTIONS
*
PNS
 OPENED FEATURE FILE WITH  HEADER
 NAME,LABL,JD, LB,IC, MV,IOPT,IHIS, FIRS, FLAS
 FEAT 1111 17  80  3 198    0    1 .50E-02 .10E+01
 OPENED CLAS FILE WITH  HEADER
  NAME  LABLC    JDX    ICX    NTC   MBUC    MKV   NENT   NCIX   ISYM  IUKER
  CLAS 111101     17     21      3     50      0     25     19      1      0
 SUBSET  CLASS=  88
 SUMMARY CONCLUSION
 .6582  .3418 0.0000
 CONFUSION MATRIX
  198  1 86  1 12
   82  2  9 47 42
  191  3 24 23 52
```

FIG 7. BOX80 ERROR RATE (FOBW SET 1)

dependent upon a full covariance matrix for each class. This is many times more expensive in computation time and in memory usage than the BOX80 algorithm. The OLPARS NMV procedure includes an option (-2) based upon an inverse weighting matrix. This is similar to the BOX80 classifier algorithm. Fig 8 shows that OLPARS' error rate using this option is virtually identical to the BOX80 error rate. Thus the BOX80 classifier is algorithmically acceptable. (Note that although BOX80, OLPARS and SPSS all allow their users options to experimentally define parameters which may decrease error rates, none were used in any of these experiments.)

(c) A second sample of vectors from the FOBW data set was processed using the BOX80 system and using the OLPARS' NMV-2 option. Figs 9 and 10 show respective error rates to be again nearly identical. Note, however, the over ten percent increase in the error rate for this sample over that for the previous sample. This is simply due to differences in the data collected for each sample. The overall data set was not analyzed to deliberately extract a worst-case subset. This leads to a rhetorical argument which is presented as an aside. Assume that this second sample was actually the initial sample. Allow that it was accepted as the design test-bed. Consider the development and usage costs for the software for both iterative generation of a class defining structure, and for implementation of the classifier. Would implementation of an optimal piecewise linear hyperplane be justified?

```
NMV
\THE CURRENT LOGIC NODE IS 1
\ENTER AN OPTION:

 1  \SIMPLE NEAREST MEAN VECTOR
 2  \INVERSE VARIANCE WEIGHTING (WEIGHTING VECTOR)
 3  \MAHALANOBIS (WEIGHTING MATRIX)
2
\DO YOU WISH TO IMPLEMENT ANY REJECT BOUNDARIES?
NO
\PARTIAL \NEAREST \MEAN \VECTOR \EVALUATION:   AFITUN14 ****
\NUMBER OF DIMENSIONS = 14


     TRUE CLASS

     \A\A\A\A \B\B\B\B \C\C\C\C
\A\A\A\A  164    4   41
\B\B\B\B    1   41   55
\C\C\C\C   34   36  111
REJT     0    0    0

TOTL  199   81  207
CORR  164   41  111
%COR 82.4 50.6 53.6
EROR   35   40   96
%ERR 17.6 49.4 46.4
REJT     0    0    0
%REJ  0.0  0.0  0.0


TOTAL NUMBER OF VECTORS = 487
OVERALL CORRECT    316 FOR  64.89%
OVERALL ERROR      171 FOR  35.11%
OVERALL REJECT       0 FOR   0.00%
 \DO YOU WANT A HARD COPY OF THIS MATRIX?
NO
\DO YOU WISH TO CHANGE THE WEIGHTING, OR ANY REJECT VALUES?
NO
 SUMMRYCM  DISPLACM  HRDCPYCM  NMVMOD    \CURRENT \OPTION: NMV
****     R 1500 4.221 5.574 194 LEVEL 3, 26

SUMMRYCM
\PARTIAL \NEAREST \MEAN \EVALUATION \SUMMARY:   AFITUN14 ****
\NUMBER OF DIMENSIONS = 14

NODE    %C     %E     %R

\A\A\A\A 82.41 17.59  0.00
\B\B\B\B 50.62 49.38  0.00
\C\C\C\C 53.62 46.38  0.00
 OVERALL CORRECT
      64.89%
 OVERALL ERROR
      35.11%
 OVERALL REJECT
       0.00%
```

FIG 8. OLPARS NMV-2 ERROR RATE

```
 TRYOUT
 ENTER OPTIONS
*PNFS

 FEAT FILE    1000
 CLAS FILE  100001
 SUBSET
 ENTER CLASS AND DIMENSIONS
 +CL
*088

 1LOG  FIGURES OF MERIT FOR DIMENSIONS
  6= .3526E+02   7= .3060E+02  11= .2954E+02   9= .2661E+02  12= .2548E+02   8= .2370E+02   1= .2320E+02  10= .2147E+0
  3= .2083E+02  13= .2025E+02  14= .1838E+02   4= .1592E+02   2= .1301E+02   5= .1258E+02
 2SUM  FIGURES OF MERIT FOR DIMENSIONS
  6= .2700E+00  11= .5660E-01   7= .5242E-01  12= .1392E-01   9= .1339E-01   3= .1055E-01   1= .6221E-02  13= .6092E-0
 14= .2565E-02   8= .2181E-02  10= .1305E-02   2= .1144E-02   5= .3196E-03   4= .3188E-03
 ENTER F/M SET NUMBER
*2

 SUMMARY CONCLUSION
 .5553  .4447 0.0000
 SUBSPACE TAGS
 6 11  7 12  9  3  1 13 14  8 10  2  5  4
 SUBSPACE ERROR RATES
 1=57/42  2=55/44  3=58/41  4=57/42  5=57/42  6=57/42  7=56/43  8=56/43  9=56/43 10=56/43 11=55/44 12=55/44
13=55/44 14=55/44
```

FIG 9. BOX80 ERROR RATE (FOBW SAMPLE 2)

```
Partial Nearest Mean Vector Evaluation:
Number of dimensions = 14


          true class

          AAAA BBBB CCCC
AAAA       160   12   42
BBBB        33   58  121
CCCC         6   11   44
rejt         0    0    0

totl       199   81  207
corr       160   58   44
%cor      80.4 71.6 21.3
eror        39   23  163
%err      19.6 28.4 78.7
rejt         0    0    0
%rej       0.0  0.0  0.0


total number of vectors = 487
overall correct    262 for  53.80%
overall error      225 for  46.20%
overall reject       0 for   0.00%


Overall Evaluation Summary:
Dataset kdigrams **** passed against logic designed on firshalf
Number of dimensions = 14

node    %c     %e     %r

AAAA 77.89 22.11  0.00   overall correct
BBBB 80.49 19.51  0.00        52.66%
CCCC 17.39 82.61  0.00   overall error
                              47.34%
                         overall reject
                              0.00%
```

FIG 10. OLPARS NMV-2 SAMPLE-2 ERROR RATE
FOBW DATA SET

(d) The OLPARS system offers a variety of feature evaluation algorithms. Two were used to evaluate the features of the samples discussed above. Fig 11 ranks the features on their ability to separate class pairs. Fig 12 presents overall merit at interclass discrimination and ranks features in this order. Fig 13 presents BOX80 merit figures. F/M set "1LOG" corresponds to the M_1 matrix discussed earlier; F/M set "2SUM" corresponds to the M_2 matrix. Features are ordered by descending figure of merit. It was noted that both BOX80 sets of merit figures disagree with OLPARS feature ranking. Each set of merit figures was then compared in terms of the classification errors which its use produced.

(e) As discussed earlier, the BOX80 feature subset selection process operates on a set of proper nested feature subspaces during each trial recognition of the test data set. In Figs 7, 9 and 13 the summary conclusion percentages reflect use of the complete set of 14 features in the class defining structure. The "subspace tags' list gives the order of features used in each of the nested subspaces which are evaluated. Each tag denotes the last added feature. The rates presented for each subspace are the percentage correctly classified followed by the percentage in error. The nested subspaces are first, that containing the left-most listed subspace tag, and then, that containing the left-most pair of tags, and so forth. Examination of Fig 13 shows that

```
FEATURES AFITAF14
\DO YOU WISH TO DO MEASUREMENT SELECTION INTERACTIVELY?
N=YES
\DO YOU WISH TO SELECT ANY MEASUREMENTS TO START WITH?
NO
\ENTER THE DEFAULT DISPLAY TO BE PRESENTED AT EACH ITERATION.
1  RNK$OALL
2  UN$BBCP
3  UN$BBC
1

       MEAS.    VALUE      CLASS PAIR
        ◆ 2      21.8568     \A     \C/\A
          1      20.1878     \C     \C/\A
          3      17.0796     \A     \B/\A
          6      15.5953     \A     \C/\A
         11      12.4391     \A     \C/\A
          4      11.5362     \A     \B/\A
          7       9.9862     \A     \B/\A
          8       9.0517     \A     \C/\A
         12       6.7854     \A     \B/\A
          9       6.5776     \A     \B/\A
         13       6.4375     \A     \B/\A
          5       5.2048     \A     \B/\A
         10       5.0559     \A     \C/\A
         14       3.7700     \A     \B/\A
```

FIG 11. OLPARS OVERALL FEATURE RANK

```
FEATURES AFITAF14
\DO YOU WISH TO DO MEASUREMENT SELECTION INTERACTIVELY?
YES
\DO YOU WISH TO SELECT ANY MEASUREMENTS TO START WITH?
NO
\ENTER THE DEFAULT DISPLAY TO BE PRESENTED AT EACH ITERATION.
1  RNK$OALL
2  UN$BBCP
3  UN$BBC
2

       MEAS.    VALUE
        ◆ 2      21.8568
        ◆ 1      20.1878
        ◆ 3      17.0796
          6      15.5953
         11      12.4391
          4      11.5362
          7       9.9862
          8       9.0517
         12       6.7854
          9       6.5776
         13       6.4375
          5       5.2048
         10       5.0559
         14       3.7700
```

FIG 12. OLPARS CLASS-PAIR FEATURE RANK

```
XTR1AQW,FEAT2QW,NEWC2QW

 TRYOUT
 ENTER OPTIONS
#PNF

 FEAT FILE    1111
 CLAS FILE  111101
 1LOG  FIGURES OF MERIT FOR DIMENSIONS
  6= .3740E+02   7= .3024E+02   1= .2991E+02  11= .2895E+02   9= .2628E+02  12= .2464E+02   8= .2299E+02   3= .2156E+0■
 10= .2124E+02  13= .2030E+02   2= .1870E+02  14= .1838E+02   4= .1545E+02   5= .1252E+02
 2SUM  FIGURES OF MERIT FOR DIMENSIONS
  6= .4361E+00  11= .4953E-01   7= .4767E-01   1= .3392E-01   9= .1287E-01   3= .1188E-01  12= .1169E-01   2= .5690E-0■
 13= .5501E-02  14= .2639E-02   8= .1720E-02  10= .1220E-02   5= .3536E-03   4= .2645E-03
 ENTER F/M SET NUMBER
#1

 SUMMARY CONCLUSION
 .6582  .3418 0.0000
 SUBSPACE TAGS
 6  7  1 11  9 12  8  3 10 13  2 14  4  5
 SUBSPACE ERROR RATES
 1=59/40  2=57/42  3=61/38  4=65/34  5=64/35  6=66/33  7=66/33  8=66/33  9=65/34 10=64/35 11=67/32 12=66/33
13=65/34 14=65/34
 ENTER F/M SET NUMBER
#2

 SUMMARY CONCLUSION
 .6582  .3418 0.0000
 SUBSPACE TAGS
 6 11  7  1  9  3 12  2 13 14  8 10  5  4
 SUBSPACE ERROR RATES
 1=59/40  2=65/34  3=64/35  4=65/34  5=64/35  6=64/35  7=66/33  8=69/30  9=69/30 10=66/33 11=66/33 12=66/33
13=66/33 14=65/34
```

FIG 13. BOX80 FEATURES - FIGURES OF MERIT

performance peaks at subspace 11 for F/M set "1LOG" and at subspace 9 for F/M set "2SUM". In both cases feature 2 has just been added to the subspace. Fig 14 shows BOX80 use of a user defined set of "subspace tags" which includes features 2 and 1. Again a performance peak is noted.

It has been noted that exhaustive search is the only method by which the 'best' subset of features can be found. The foregoing discussion illustrates how BOX80 algorithms can be used to guide a heuristic search which improves performance and yet is not exhaustive. It also illustrates the greater strength of OLPARS' feature evaluation algorithm. The BOX80 subset evaluation technique has no counterpart in the OLPARS system which performs each classification trial separately.

(f) Fig 15 illustrates the BOX80 procedure for recording the selection of a subset of features. The newly generated class defining structure produces an overall error rate of 28 percent. Fig 16 shows the procedure for generating scaled eight-bit data values for the microprocessor based classifier. The TRYOUT module option 'B' requests this 'byte' scaling. The zapping process referenced in the figure nullifies specified feature dimensions (i.e., those not to be used), by arbitrarily expanding the value of their respective boundaries (variances) to a large value. This is further discussed in chapter 5. In this run, a trial recognition was then accomplished using integer arithmetic.

```
 ENTER F/M SET NUMBER
*0

 SPECIFY SUBSPACE TAGS
 KK,KK,...
*01,02,03,06,08,11,99

 SUMMARY CONCLUSION
 .6582  .3418 0.0000
 SUBSPACE TAGS
 1  2  3  6  8 11  4  5  7  9 10 12 13 14
 SUBSPACE ERROR RATES
 1=61/38  2=71/28  3=70/29  4=69/30  5=69/30  6=69/30  7=69/30  8=67/32  9=67/32 10=67/32 11=67/32 12=67/32
13=67/32 14=65/34
 ENTER F/M SET NUMBER
*0

 SPECIFY SUBSPACE TAGS
 KK,KK,...
*02,06,01,03,08,11,99

 SUMMARY CONCLUSION
 .6582  .3418 0.0000
 SUBSPACE TAGS
 2  6  1  3  8 11  4  5  7  9 10 12 13 14
 SUBSPACE ERROR RATES
 1=66/33  2=70/29  3=70/29  4=69/30  5=69/30  6=69/30  7=69/30  8=67/32  9=67/32 10=67/32 11=67/32 12=67/32
13=67/32 14=65/34
 ENTER F/M SET NUMBER
*3

   QUIT TRYOUT
     STOP
      7.159 CP SECONDS EXECUTION TIME
```

FIG 14. USER SELECTED FEATURE ORDER

```
 TRYOUT
 ENTER OPTIONS
*
PNS
 OPENED FEATURE FILE WITH  HEADER
 NAME,LABL,JD, LB,IC, MV,IOPT,IHIS, FIRS, FLAS
 FEAT 1111 17  80  3 198    0    1 .50E-02 .10E+01
 OPENED CLAS FILE WITH  HEADER
  NAME  LABLC    JDX    ICX    NTC   MBUC    MKV   NENT   NCIX   ISYM  IUKER
  CLAS 111101     17     21      3     50      0     25     19      1      0
 SUBSET  CLASS=  99
 DD,DD,...
*
  1  2 99  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   SUBSET
   ZAPS=   0  0  3  4  5  6  7  8  9 10 11 12 13 14
 SUMMARY CONCLUSION
 .7134  .2866 0.0000
 CONFUSION MATRIX
  198  1 86  1 12
   82  2  3 41 54
  191  3 16 15 68
 SUBSET  CLASS=   0
```

FIG 15. ERROR RATE - SELECTED FEATURE SUBSET

```
 TRYOUT
 ENTER OPTIONS
*
PBS
 OPENED FEATURE FILE WITH  HEADER
 NAME,LABL,JD, LB,IC, MV,IOPT,IHIS, FIRS, FLAS
 FEAT 1111 17  80  3 198    0    1 .50E-02 .10E+01
 OPENED CLAS FILE WITH  HEADER
  NAME  LABLC    JDX    ICX    NTC   MBUC    MKV   NENT   NCIX   ISYM  IUKER
  CLAS 111101     17     21      3     50      0     25     19      1      0
 SUBSET  CLASS=  99
 DD,DD,...
*
  1  2 99  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   SUBSET
   ZAPS=   0  0  3  4  5  6  7  8  9 10 11 12 13 14
 SUMMARY CONCLUSION
 .6985  .3015 0.0000
 CONFUSION MATRIX
  198  1 87  1 11
   82  2  3 37 58
  191  3 16 17 65
 SUBSET  CLASS=   0
```

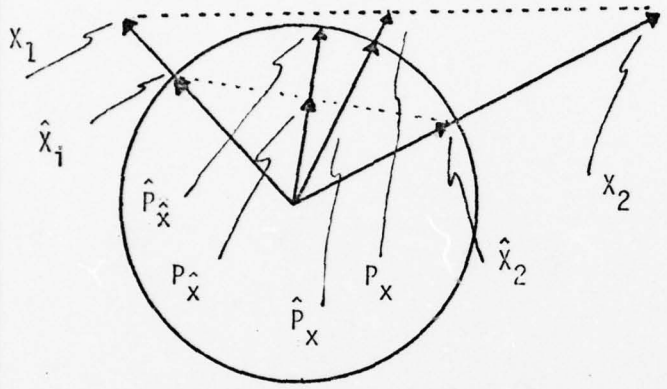
FIG 16. ERROR RATE - BYTE-SCALED COMPONENTS

This simulates the byte valued operations actually performed in the microprocessor. Error rate increases by only 1.5 percent and remains below both the error rate achieved by OLPARS(NMV-2) and that produced by BOX80 on the original 14 component data set. From these facts, BOX80 procedures for subset selection, and for generation of the class-defining structure, are judged acceptable.

(2) Figs 17 through 23 apply to Fourier transformed alphabetic data. This data set consists of 3900 feature vectors of 49 components each. The components of these vectors are the real and imaginary parts of complex numbers. These numbers are output by low frequency filtered Fourier transforms of two space images of digitized letters. The technique used to produce these vectors has been discussed in several AFIT theses (Ref 14, 31) as well as in the as yet unpublished work by Sponaugle (Ref 33). These vectors form a 26-class problem. Programs produced by Sponaugle were used to establish benchmark error rates for classification of this data.

(a) The components of the vectors in this data set were assumed to be largely uncorrelated because they had been generated by an orthogonal linear transform. The use of both real and imaginary parts of the values output by this transform suggests the caveat 'largely' since the transform produces orthogonal complex values. The size of the data set precluded use of SPSS to generate correlation indices as was done with the FOBW data.

DATA TRANSFORMS:



Notation:

X = feature vector

$\hat{X}$ = unit vector
$= X/|X|$

P_X = mean x vector
$= (\sum_{i=1}^{N} x_i)/(N)$

$P_{\hat{X}}$ = mean $\hat{X}$ vector

$\hat{P}_X$ = unit mean X vector
$= P_X/|P_{\hat{X}}|$

$\hat{P}_{\hat{X}}$ = unit mean $\hat{X}$ vector
$= P_{\hat{X}}/|P_{\hat{X}}|$

RESULTS:

		Overall Error Rate	No. 'perfect' alphabets
A.	x_i vs P_X -	18.41	8
B.	$\hat{X}_i$ vs $P_{\hat{X}}$ -	11.50	8
C.	$\hat{X}_i$ vs $\hat{P}_{\hat{X}}$ -	11.20	8
D.	$\hat{X}_i$ vs $\hat{P}_X$ -	10.88	8
E.	$\hat{X}_i$ vs $P_{\hat{X}}$ and Σ -	7.11	36

where Σ such that for all $\sigma_{ii} \in \Sigma$, $\sigma_{ii} = [\mathrm{var}(P_{\hat{X}}; \hat{X})]^{\frac{1}{2}}$

Fig. 17. Alphabet Classification Experiments

(b) This data set was processed using a classification program written by Sponaugle. The program uses a minimum distance algorithm. It produces an overall error rate, a confusion matrix and individual error rates for each alphabet. Appendix L records output from this program which is summarized in Fig 17. Sponaugle's work included heuristic experimentation which attempted to establish appropriate normalizing transforms with which to precondition the feature vectors. The original data (after application of centering algorithms to the data input to the Fourrier transform), classified with an error rate of 18.4 percent. Arguing that "thick" letters would in general have larger vector magnitudes than "thin" letters, as is shown diagrammatically by vectors X_1 and X_2, Sponaugle normalized the feature vectors by their magnitudes and again classified the data. His least error rate was produced by experiment D. The intuitively difficult combination of $\hat{X}_i$ and $\hat{P}_x$ in this experiment may be explained by the hypothesis that this normalization retains the angular variation implicit in the original data vectors while standardizing vector magnitudes. The BOX80 classifier algorithm was integrated into this minimum distance classifier. A trial classification produced the 7.1 percent error rate reported in the figure under item E. The decrease in error rate, and the significant increase in the count of alphabetic fonts recognized as identical, qualifies the BOX80 classifier as significant. For reference by future AFIT experiments, the identically recognized alphabetic fonts are recorded in Table II.

TABLE II

Identically Recognized Alphabets

Experiment A:

28, 48, 104, 139, 16, 33, 35, 41

Experiment B:

28, 9, 10, 139, 16, 33, 35, 75

Experiment C:

28, 9, 10, 139, 16, 33, 35, 75

Experiment D:

28, 9, 10, 139, 16, 33, 35, 75

Experiment E:

28, 9, 10, -, -, 33, 35, 75, 8, 15

19, 26, 30, 32, 25, 27, 29, 48, 50, 58

104, 127, 129, 133, 143, 41, 51, 66, 83, 90,

103, 108, 116, 140, 144, 149, 150

(c) The BOX80 system was used to process a 780 vector subset of this alphabetic data. A subset was used only to reduce process time; it does not affect the validity of this benchmark. A confusion matrix for this process is shown in Fig 18, with an overall error rate of 4.6 percent. The decrease in error rate appears to correlate with the fact that the 30-letter sample per class used in this experiment included 10 of the "identical" alphabetic fonts reported in Table II. This experiment is significantly different from that reported above in one important respect. As noted under "system efficiency" in this section, the BOX80 classifier used less than 55K of memory and 23 cpu seconds for its operation. However, the alphabet classifier required 140K of memory and 205 cpu seconds to complete a trial classification run. After scaling this execution time by the reduced size of the BOX80 data sample, a 2:1 throughput increase is still indicated. The minimal BOX80 memory use results from its efficient data structures. This contrasts to the far greater memory requirement of the alphabetic classifier. It should be noted that the alphabetic classifier accumulates and stores extensive statistics for output; these account for part of its memory requirement. The classification rate presented for this set of 49 component alphabetic feature vectors correlates well with Tallman's simulated result, 95.80 (Ref 35:86).

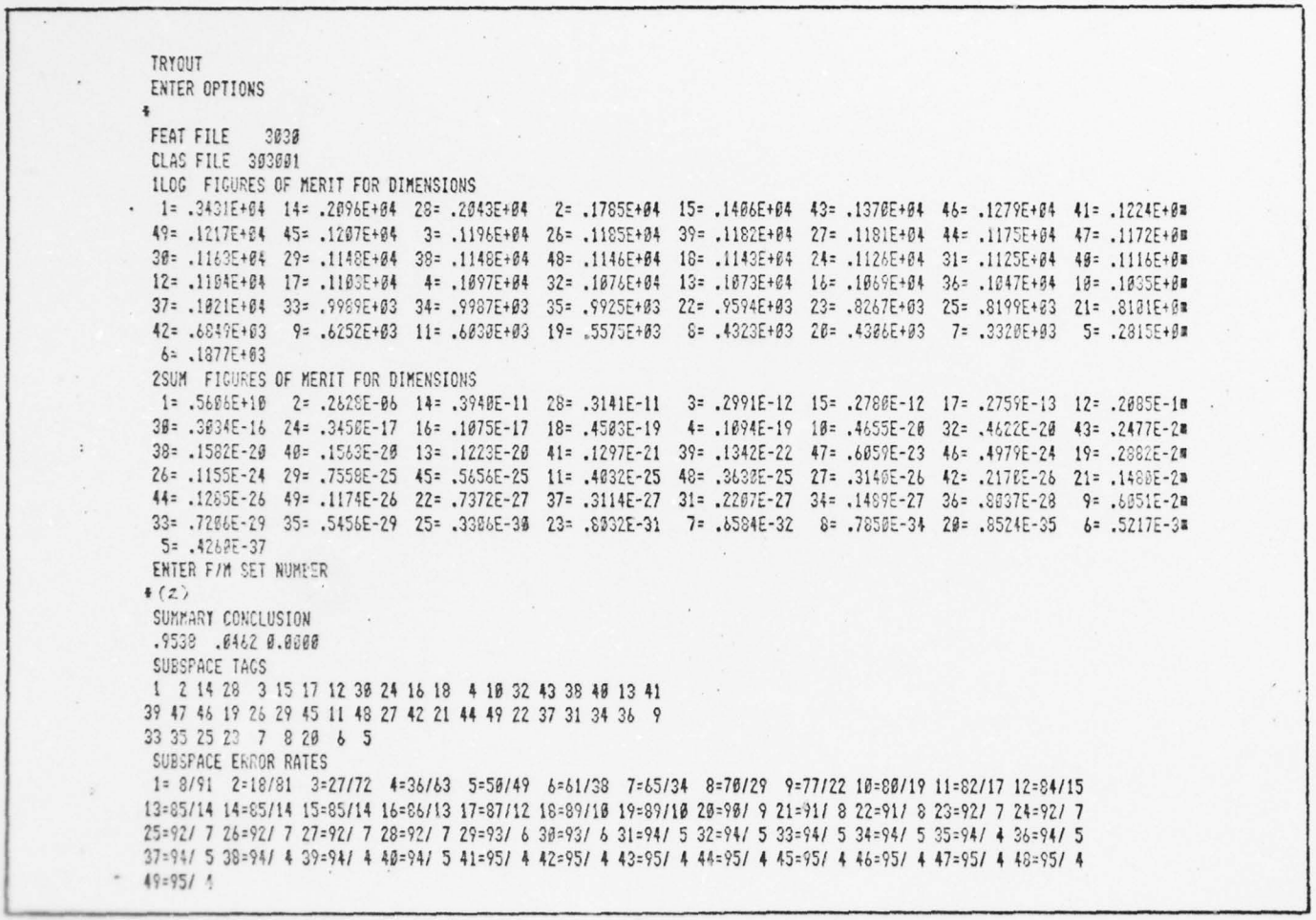
(d) Figs 19 through 21 show BOX80 merit figures computed for this 49 component alphabetic data set. Notice that

```
 TRYOUT
 ENTER OPTIONS
+
PNS
 OPENED FEATURE FILE WITH  HEADER
 NAME,LABL,JD, LB,IC, MV,IOPT,IHIS, FIRS, FLAS
 FEAT 3030 52  30 26  30    3    1-.10E+01 .10E+01
 OPENED CLAS FILE WITH  HEADER
  NAME  LABLC    JDX    ICX    NTC   MBUC    MKV   NENT   NCIX   ISYM  IUKER
  CLAS 303001     52     85     26     20      0    104     82      1      0
 SUBSET  CLASS=  88
 SUMMARY CONCLUSION
 .9538  .0462 0.0000
 CONFUSION MATRIX
   30  1 93  0  0  0  0  0  0  0  0  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  2  0 93  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0
   30  3  0  0 96  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  4  0  0  0100  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  5  0  0  3  0 93  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  6  0  0  0  0  0 96  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  7  0  0  3  0  0  0 93  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0
   30  8  0  0  0  0  0  0  0 96  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  9  0  0  0  0  0  0  0  0 96  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30 10  0  0  0  0  0  0  0  0  0100  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30 11  0  0  0  0  0  0  0  0  0  0 96  0  0  0  0  0  0  0  3  0  0  0  0  0  0
   30 12  0  0  0  0  0  0  0  0  0  3  0 96  0  0  0  0  0  0  0  0  0  0  0  0  0
   30 13  0  0  0  0  0  0  0  0  0  0  0  0 96  0  0  0  0  0  0  0  0  0  3  0  0  0
   30 14  0  0  0  0  0  0  0  3  0  0  0  0  6 89  0  0  0  0  0  0  0  0  0  0  0
   30 15  0  0  0  3  0  0  0  0  0  0  0  0  0  0 76  0 19  0  0  0  0  0  0  0  0
   30 16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0100  0  0  0  0  0  0  0  0  0
   30 17  0  0  3  0  0  0  0  0  0  0  0  0  0  0  3  0 93  0  0  0  0  0  0  0  0
   30 18  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  6  0  0100  0  0  0  0  0  0
   30 19  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0100  0  0  0  0  0
   30 20  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0 96  0  0  0  0  0
   30 21  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0100  0  0  0  0
   30 22  0  0  0  0  0  0  0  0  0  3  0  0  3  0  0  0  0  0  0  0  0 89  0  0  3  0
   30 23  0  0  0  0  0  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  6 89  0  0  0
   30 24  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0100  0  0
   30 25  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  3  0  0 96  0
   30 26  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0 96
```

FIG 18. ERROR RATE FOR 49 ALPHABETIC FEATURES

```
 TRYOUT
 ENTER OPTIONS
*
 FEAT FILE    3030
 CLAS FILE  303001
 1LOG  FIGURES OF MERIT FOR DIMENSIONS
  1= .3431E+04  14= .2096E+04  28= .2043E+04   2= .1785E+04  15= .1406E+04  43= .1370E+04  46= .1279E+04  41= .1224E+04
 49= .1217E+04  45= .1207E+04   3= .1196E+04  26= .1185E+04  39= .1182E+04  27= .1181E+04  44= .1175E+04  47= .1172E+04
 30= .1163E+04  29= .1148E+04  38= .1148E+04  48= .1146E+04  18= .1143E+04  24= .1126E+04  31= .1125E+04  40= .1116E+04
 12= .1104E+04  17= .1103E+04   4= .1097E+04  32= .1076E+04  13= .1073E+04  16= .1069E+04  36= .1047E+04  10= .1035E+04
 37= .1021E+04  33= .9989E+03  34= .9987E+03  35= .9925E+03  22= .9594E+03  23= .8267E+03  25= .8199E+03  21= .8101E+03
 42= .6849E+03   9= .6252E+03  11= .6030E+03  19= .5575E+03   8= .4323E+03  20= .4306E+03   7= .3320E+03   5= .2815E+03
  6= .1877E+03
 2SUM  FIGURES OF MERIT FOR DIMENSIONS
  1= .5606E+10   2= .2628E-06  14= .3940E-11  28= .3141E-11   3= .2991E-12  15= .2780E-12  17= .2759E-13  12= .2085E-14
 30= .3034E-16  24= .3450E-17  16= .1075E-17  18= .4503E-19   4= .1094E-19  10= .4655E-20  32= .4622E-20  43= .2477E-20
 38= .1582E-20  40= .1563E-20  13= .1223E-20  41= .1297E-21  39= .1342E-22  47= .6059E-23  46= .4979E-24  19= .2882E-24
 26= .1155E-24  29= .7558E-25  45= .5656E-25  11= .4032E-25  48= .3630E-25  27= .3140E-26  42= .2170E-26  21= .1490E-26
 44= .1285E-26  49= .1174E-26  22= .7372E-27  37= .3114E-27  31= .2207E-27  34= .1489E-27  36= .8037E-28   9= .6051E-28
 33= .7206E-29  35= .5456E-29  25= .3306E-30  23= .8032E-31   7= .6584E-32   8= .7850E-34  20= .8524E-35   6= .5217E-37
  5= .4260E-37
 ENTER F/M SET NUMBER
*(1)
 SUMMARY CONCLUSION
 .9538  .0462 0.0000
 SUBSPACE TAGS
 1 14 28  2 15 43 46 41 49 45  3 26 39 27 44 47 30 29 38 48
18 24 31 40 12 17  4 32 13 16 36 10 37 33 34 35 22 23 25 21
42  9 11 19  8 20  7  5  6
 SUBSPACE ERROR RATES
 1= 8/91  2=14/85  3=24/75  4=36/63  5=50/49  6=54/45  7=60/39  8=63/36  9=64/35 10=66/33 11=73/26 12=80/19
13=81/18 14=83/16 15=84/14 16=85/14 17=86/13 18=87/12 19=88/11 20=88/11 21=90/ 9 22=91/ 8 23=90/ 9 24=91/ 8
25=91/ 8 26=92/ 7 27=92/ 7 28=92/ 7 29=92/ 7 30=93/ 6 31=93/ 6 32=93/ 6 33=93/ 6 34=93/ 6 35=94/ 5 36=94/ 5
37=94/ 5 38=94/ 5 39=94/ 5 40=94/ 5 41=94/ 5 42=94/ 5 43=95/ 4 44=95/ 4 45=95/ 4 46=95/ 4 47=95/ 4 48=95/ 4
49=95/ 4
```

FIG 19. ERROR RATE FOR MERIT 1 SUBSPACES



```
TRYOUT
ENTER OPTIONS
*
FEAT FILE   3030
CLAS FILE  303001
1LOG  FIGURES OF MERIT FOR DIMENSIONS
 1= .3431E+04  14= .2096E+04  28= .2043E+04   2= .1785E+04  15= .1406E+04  43= .1370E+04  46= .1279E+04  41= .1224E+04
49= .1217E+04  45= .1207E+04   3= .1196E+04  26= .1185E+04  39= .1182E+04  27= .1181E+04  44= .1175E+04  47= .1172E+04
30= .1163E+04  29= .1148E+04  38= .1148E+04  48= .1146E+04  18= .1143E+04  24= .1126E+04  31= .1125E+04  40= .1116E+04
12= .1104E+04  17= .1103E+04   4= .1097E+04  32= .1076E+04  13= .1073E+04  16= .1069E+04  36= .1047E+04  10= .1035E+04
37= .1021E+04  33= .9989E+03  34= .9987E+03  35= .9925E+03  22= .9594E+03  23= .8267E+03  25= .8199E+03  21= .8101E+04
42= .6849E+03   9= .6252E+03  11= .6030E+03  19= .5575E+03   8= .4323E+03  20= .4306E+03   7= .3320E+03   5= .2815E+04
 6= .1877E+03
2SUM  FIGURES OF MERIT FOR DIMENSIONS
 1= .5606E+10   2= .2628E-06  14= .3940E-11  28= .3141E-11   3= .2991E-12  15= .2780E-12  17= .2759E-13  12= .2085E-14
30= .3034E-16  24= .3450E-17  16= .1075E-17  18= .4503E-19   4= .1094E-19  10= .4655E-20  32= .4622E-20  43= .2477E-21
38= .1582E-20  40= .1563E-20  13= .1223E-20  41= .1297E-21  39= .1342E-22  47= .6059E-23  46= .4979E-24  19= .2862E-24
26= .1155E-24  29= .7558E-25  45= .5656E-25  11= .4032E-25  48= .3638E-25  27= .3140E-26  42= .2170E-26  21= .1480E-24
44= .1265E-26  49= .1174E-26  22= .7372E-27  37= .3114E-27  31= .2207E-27  34= .1489E-27  36= .8037E-28   9= .6051E-24
33= .7206E-29  35= .5456E-29  25= .3306E-30  23= .6032E-31   7= .6584E-32   8= .7850E-34  20= .8524E-35   6= .5217E-34
 5= .4260E-37
ENTER F/M SET NUMBER
*(z)
SUMMARY CONCLUSION
.9538  .0462 0.0000
SUBSPACE TAGS
 1  2 14 28  3 15 17 12 30 24 16 18  4 10 32 43 38 40 13 41
39 47 46 19 26 29 45 11 48 27 42 21 44 49 22 37 31 34 36  9
33 35 25 23  7  8 20  6  5
SUBSPACE ERROR RATES
 1= 8/91  2=18/81  3=27/72  4=36/63  5=50/49  6=61/38  7=65/34  8=70/29  9=77/22 10=80/19 11=82/17 12=84/15
13=85/14 14=85/14 15=85/14 16=86/13 17=87/12 18=89/10 19=89/10 20=90/ 9 21=91/ 8 22=91/ 8 23=92/ 7 24=92/ 7
25=92/ 7 26=92/ 7 27=92/ 7 28=92/ 7 29=93/ 6 30=93/ 6 31=94/ 5 32=94/ 5 33=94/ 5 34=94/ 5 35=94/ 4 36=94/ 5
37=94/ 5 38=94/ 4 39=94/ 4 40=94/ 5 41=95/ 4 42=95/ 4 43=95/ 4 44=95/ 4 45=95/ 4 46=95/ 4 47=95/ 4 48=95/ 4
49=95/ 1
```

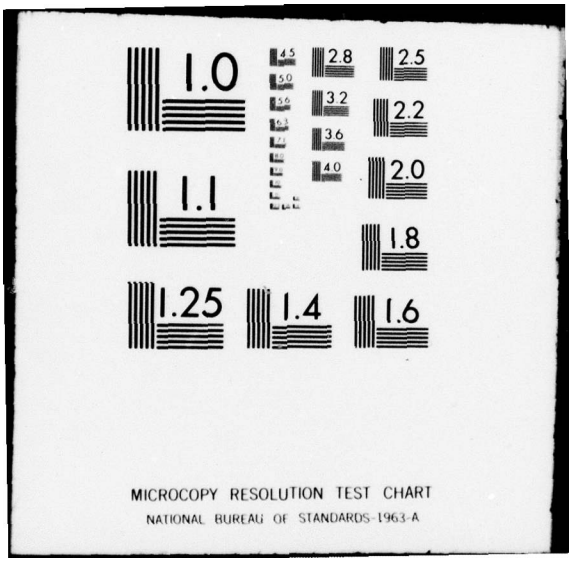
FIG 20. ERROR RATES FOR MERIT-2 SUBSPACES

AD-A064 194 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 5/8
A DEVELOPMENT SYSTEM FOR MICROPROCESSOR BASED PATTERN RECOGNIZE--ETC(U)
DEC 78 J R LEARY

UNCLASSIFIED AFIT/GCS/EE/78-12-VOL-1 NL

2 OF 2
AD
A064194

END
DATE FILMED
4-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
TRYOUT
ENTER OPTIONS
*
FEAT FILE    3030
CLAS FILE  303001
ILOG  FIGURES OF MERIT FOR DIMENSIONS
 1= .3431E+04  14= .2096E+04  28= .2043E+04   2= .1785E+04  15= .1406E+04  43= .1370E+04  46= .1279E+04  41= .1224E+04
49= .1217E+04  45= .1207E+04   3= .1196E+04  26= .1185E+04  39= .1182E+04  27= .1181E+04  44= .1175E+04  47= .1172E+04
30= .1163E+04  29= .1148E+04  38= .1148E+04  48= .1146E+04  18= .1143E+04  24= .1126E+04  31= .1125E+04  40= .1116E+04
12= .1104E+04  17= .1103E+04   4= .1097E+04  32= .1076E+04  13= .1073E+04  16= .1069E+04  36= .1047E+04  10= .1035E+04
37= .1021E+04  33= .9989E+03  34= .9987E+03  35= .9925E+03  22= .9594E+03  23= .8267E+03  25= .8199E+03  21= .8101E+03
42= .6849E+03   9= .6252E+03  11= .6030E+03  19= .5575E+03   8= .4323E+03  20= .4306E+03   7= .3320E+03   5= .2815E+03
 6= .1877E+03
2SUM  FIGURES OF MERIT FOR DIMENSIONS
 1= .5606E+10   2= .2628E-06  14= .3940E-11  28= .3141E-11   3= .2991E-12  15= .2780E-12  17= .2759E-13  12= .2085E-14
30= .3034E-16  24= .3450E-17  16= .1075E-17  18= .4503E-19   4= .1094E-19  10= .4655E-20  32= .4622E-20  43= .2477E-20
38= .1582E-20  40= .1563E-20  13= .1223E-20  41= .1297E-21  39= .1342E-22  47= .6059E-23  46= .4979E-24  19= .2882E-24
26= .1155E-24  29= .7558E-25  45= .5656E-25  11= .4032E-25  48= .3630E-25  27= .3140E-26  42= .2170E-26  21= .1480E-26
44= .1285E-26  49= .1174E-26  22= .7372E-27  37= .3114E-27  31= .2207E-27  34= .1489E-27  36= .8037E-28   9= .6051E-28
33= .7206E-29  35= .5456E-29  25= .3306E-30  23= .8032E-31   7= .6584E-32   8= .7858E-34  20= .8524E-35   6= .5217E-37
 5= .4260E-37
ENTER F/M SET NUMBER
* (0)
SPECIFY SUBSPACE TAGS
KK,KK,...
*
SUMMARY CONCLUSION
.9538  .0462 0.0000
SUBSPACE TAGS
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49
SUBSPACE ERROR RATES
 1= 8/91  2=18/81  3=38/61  4=45/54  5=46/53  6=49/50  7=50/49  8=56/43  9=61/38 10=67/32 11=69/30 12=71/28
13=73/26 14=74/25 15=78/21 16=78/21 17=79/20 18=80/19 19=81/18 20=81/18 21=82/17 22=84/15 23=84/15 24=88/11
25=88/11 26=89/10 27=89/10 28=89/ 9 29=90/ 9 30=91/ 8 31=91/ 8 32=91/ 8 33=91/ 8 34=91/ 8 35=92/ 7 36=92/ 7
37=92/ 7 38=92/ 7 39=93/ 6 40=93/ 6 41=94/ 5 42=94/ 5 43=94/ 4 44=94/ 5 45=94/ 5 46=94/ 4 47=94/ 4 48=95/ 4
49=95/ 4
ENTER F/M SET NUMBER
*
  QUIT TRYOUT
```

FIG 21. ERROR RATES FOR ARBITRARY FEATURE SUBSPACES

subspace error rates all decrease as the number of subspace features increases. At subspace 20 the error rates are 11, 9, and 18 percent for merit figures 1, 2, and 0, respectively. (The 0 set consists of an arbitrary 49 components by order of increasing dimension.) These error rates support two conclusions. First, the BOX80 system outperforms the benchmark in both error rate and number of features. Second, F/M set 2 has the lowest error rate and is the more robust of the two figures of merit. This agrees with the analysis reported for the FOBW data set.

(e) Fig 22 presents confusion matrices and overall error rates for feature subspace 20 from F/M set 2. The majority of the errors are concentrated in separating classes 15/17 and 22/23. These classes represent the letters O and Q and the letters V and W which are readily confused by printed noise.

(f) Fig 23 shows, again via simulation, an overall error rate and a confusion matrix for byte sealed component values. It indicates that the BOX80 system development hypothesis is justified. That is, with a minimized use of memory, and the BOX80 classifier, an acceptable error rate can be attained.

Acceptability. The term "acceptable" has been used freely in the foregoing discussion. From its last use, in the context of all foregoing discussion, a precise meaning can be inferred. Acceptability is a complex function of cost and benefit. However, it is a relative term which implies not only that resources meet

```
 TRYOUT
 ENTER OPTIONS
*
PBS
 OPENED FEATURE FILE WITH  HEADER
 NAME,LABL,JD, LB,IC, MV,IOPT,IHIS, FIRS, FLAS
 FEAT 3030 52  30 26  30    3    1-.10E+01 .10E+01
 OPENED CLAS FILE WITH  HEADER
  NAME  LABLC    JDX    ICX    NTC    MBUC    MKV   NENT   NCIX   ISYM  IUKER
  CLAS 303001     52     85     26     20      0    104     82      1      0
 SUBSET  CLASS=  99
 DD,DD,...
*
  1  2 14 28  3 15 17 12 30 24 16 18  4 10 32 43 38 40 13 41
 DD,DD,...
*
 99  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   SUBSET
   ZAPS=   0  0  0  0  5  6  7  8  9  0 11  0  0  0  0  0  0  0 19 20
21 22 23  0 25 26 27  0 29  0 31  0 33 34 35 36 37  0 39  0
 0 42  0 44 45 46 47 48 49
 SUMMARY CONCLUSION
 .9077  .0923 0.0000
 CONFUSION MATRIX
   30  1 86  0  0  0  0  0  0  0  0 13  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  2  0 83  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  9  3  0  0  0  0  0  0
   30  3  0  0 96  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  4  0  0  3 93  0  0  0  0  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0
   30  5  0  0  0  0 96  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  6  0  0  0  0  3 86  0  0  0  3  0  0  0  0  0  6  0  0  0  0  0  0  0  0  0
   30  7  0  0  6  3  0  0 86  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0
   30  8  0  0  0  0  0  0  0 86  0  6  0  0  3  0  0  0  0  3  0  0  0  0  0  0  0
   30  9  0  0  0  3  0  0  0  0 89  0  3  0  0  0  0  0  0  0  0  0  3  0  0  0  0
   30 10  0  0  0  0  0  0  0  0  0100  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30 11  0  3  0  0  0  0  0  0  0  0 93  0  0  0  0  0  0  3  0  0  0  0  0  0  0
   30 12  0  0  0  0  0  0  0  0  0  3  0 96  0  0  0  0  0  0  0  0  0  0  0  0  0
   30 13  0  0  0  0  0  0  0  0  0  0  0  0 89  3  0  0  0  0  0  0  3  3  0  0  0
   30 14  0  0  0  0  0  0  0  0  0  0  0  0  9 86  0  0  0  0  3  0  0  0  0  0  0
   30 15  0  0  0  0  0  0  3  0  0  0  0  0  0  0 73  0 19  0  0  0  0  3  0  0  0
   30 16  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0 96  0  0  0  0  0  0  0  0  0
   30 17  0  0  0  6  0  0  0  0  0  3  0  0  0  0  3  0 86  0  0  0  0  0  0  0  0
   30 18  0  6  0  3  0  0  0  0  0  0  3  0  0  0  0  0  0 86  0  0  0  0  0  0  0
   30 19  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0100  0  0  0  0  0  0
   30 20  0  0  0  0  0  0  0  0  3  3  0  0  0  0  0  0  0  0  0 93  0  0  0  0  0
   30 21  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 96  3  0  0  0
   30 22  0  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0 89  6  0  0
   30 23  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 26 73  0  0
   30 24  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 0100  0
   30 25  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  6  0  0 93
   30 26  0  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

FIG 22. SUBSPACE 20 ERROR RATE(BYTE-SCALED)

```
 TRYOUT
 ENTER OPTIONS
+
PBS
 OPENED FEATURE FILE WITH  HEADER
 NAME,LABL,JD, LB,IC, MV,IOPT,IHIS, FIRS, FLAS
 FEAT 3030 52  30 26  30    3    1-.10E+01 .10E+01
 OPENED CLAS FILE WITH  HEADER
  NAME  LABLC    JDX    ICX    NTC   MBUC    MKV   NENT   NCIX   ISYM  IUKER
  CLAS 303001     52     85     26     20      0    104     82      1      0
 SUBSET  CLASS=  88
 SUMMARY CONCLUSION
 .9551  .0449 0.0000
 CONFUSION MATRIX
   30  1 93  0  0  0  0  0  0  0  0  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  2  0100  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  3  0  0 96  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  4  0  0  0100  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  5  0  0  3  0 96  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  6  0  0  0  0  0 96  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30  7  0  0  3  0  0  0 93  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0
   30  8  0  0  0  0  0  0  0 93  0  3  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0
   30  9  0  0  0  0  0  0  0  0 96  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30 10  0  0  0  0  0  0  0  0  0100  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30 11  0  0  0  0  0  0  0  0  0  0 96  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0
   30 12  0  0  0  0  0  0  0  0  0  3  0 96  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   30 13  0  0  0  0  0  0  0  0  0  0  0  0 96  0  0  0  0  0  0  0  0  0  3  0  0  0
   30 14  0  0  0  0  0  0  0  0  3  0  0  0  0  3 93  0  0  0  0  0  0  0  0  0  0  0
   30 15  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0 69  0 26  0  0  0  0  0  0  0  0
   30 16  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0100  0  0  0  0  0  0  0  0  0
   30 17  0  0  3  0  0  0  0  0  0  0  0  0  0  0  3  0 93  0  0  0  0  0  0  0  0  0
   30 18  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0100  0  0  0  0  0  0  0  0
   30 19  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0100  0  0  0  0  0  0  0
   30 20  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0 96  0  0  0  0  0  0
   30 21  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0100  0  0  0  0  0
   30 22  0  0  0  0  0  0  0  0  0  3  0  0  3  0  0  0  0  0  0  0  0 89  0  0  3  0
   30 23  0  0  0  0  0  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  6 89  0  0  0
   30 24  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0100  0  0
   30 25  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0 96  0
   30 26  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 96
```

FIG 23. SUBSPACE 49 ERROR RATE(BYTE-SCALED)

costs and benefits satisfy requirements, but also that a value judgment has been made for each case. This is why no one definition was given.

Testing Procedures

In implementing the BOX80 system, testing was a continuing process. Techniques varied with the routine or function being tested. These are indicated below.

In each module the data processing flow was evaluated by a trace at subroutine exit. Single entry, single exit subroutine paths and selective output to either the journal file or the terminal made this technique effective. Data buffer dumps were obtained from file generation processes to verify input structure and content. To simplify verification of all modules, the basic utility routines were independently tested. This procedure was not followed for support routines unique to each module because of the overhead cost for testing drivers. Finally, a simulator, INTERP80 (Ref 15) was used to exercise the data processing operations of the classifier module.

Computational code was verified by spot-checked hand calculations, analyses for self-consistency, and comparisons with known values. In the latter case, benchmark testing provided comparison values. Output from these benchmarks included statistics produced via the Statistical Package for the Social Sciences

(SPSS), feature selections identified by the On Line Pattern Analysis and Recognition System (OLPARS), and classification decisions obtained from specially written pilot routines. Finally, a trivial data set was used to verify the computations within the microprocessor classifier module.

Function options were verified by an attempt at exhaustive testing. For each option, output values were examined, and file and module interfaces were checked.

Several special tests were used. Graphics routines were deliverately passed invalid data to verify program continuity; there were no unexpected hang-ups. Feature selections were input to feature subset procedures and used in performance measurements. Finally, data from two disparate data sets were processed with the system. Thus, memory allocation algorithms and other adjustments for number of classes and dimensions were checked.

V. Design

This chapter presents the design of the BOX80 system. The flow of data through the system, processing techniques and routines, and system data structures are discussed in the first three sections. The final sections document the design of system modules.

Data Flow

The functions of the BOX80 system separate into two broad groups. To one group are assigned functions dealing with the evaluation of feature data and the generation of class definitions. The other group contains the microprocessor-based classification function. This separation conforms to the functional analysis of data flow presented in Chapter 3. The system is thus implemented in two segments of program code. Each consists of independent program modules which interact through standard data files.

Interpreter Segment. This segment consists of four independent modules whose functions allow the user to examine his feature data and to produce a standard set of class definitions. These definitions are the primary product of the interpreter segment. They link this segment to the second segment. The four modules of this segment are named CREATE, DEFINE, TRYOUT,

and FORMAT. These names reflect their basic functions.

The flow of data through the Interpreter Segment is in a circular path. Segment modules are executed by the user in an iterative cycle. The cycle ends when the user is satisfied with the simulation of classifier performance which is documented by the TRYOUT module. At this point, the classifier error rate should be acceptably low. In each iteration a file of pattern class definitions is produced. Execution of the FORMAT module can transform this data structure into one which will interface with the Classifier Segment. This is the final step in the interpretation process.

Classifier Segment. This segment consists of two independent modules. One functions as a data input routine. It allows the user to enter class defining data into microprocessor memory. The second module is a pilot model of pattern classifier which can be used in the user's system. It processes a buffer of feature vectors against a block of class definitions and outputs a classification decision for each vector. The modules in this segment are known as TAPEIN and DECIDE.

The Classifier Segment is intended as a test-bed with which to exercise a classifier module which has been configured to satisfy a user system. In such a system, a distributed process would implement the user's pattern recognition function. One microprocessor, operating in master mode, would perform the

analog to digital conversions, feature extractions, and transforms necessary to generate a feature vector for a given pattern. This microprocessor would interrupt a slave processor to store each feature vector in a RAM memory buffer accessible to the slave. The slave processor would continuously operate on the contents of this buffer, producing as output a log of classification decisions. The BOX80 system Classifier Segment illustrates this design concept by demonstrating a classifier program which can be used in the slave microprocessor. The data formats and program code for this slave processor's software are a version of the Classifier segment's DECIDE module.

The flow of data through the Interpreter and Classifier segments of the BOX80 system can be visualized as a straight line path. At execution of system modules along this path various data files are created. Files, in general, are not updated. Rather, new files are created based upon the user's analytical judgment. Any part of this path can be repeated. Thus the BOX80 system data flow supports iterative development of the classification data structure upon which the user's pattern recognizer is based. This flow is illustrated in Figure 24. Names of the modules and routines of the BOX80 system which implement this flow are listed in table III. These names are defined in table IV.

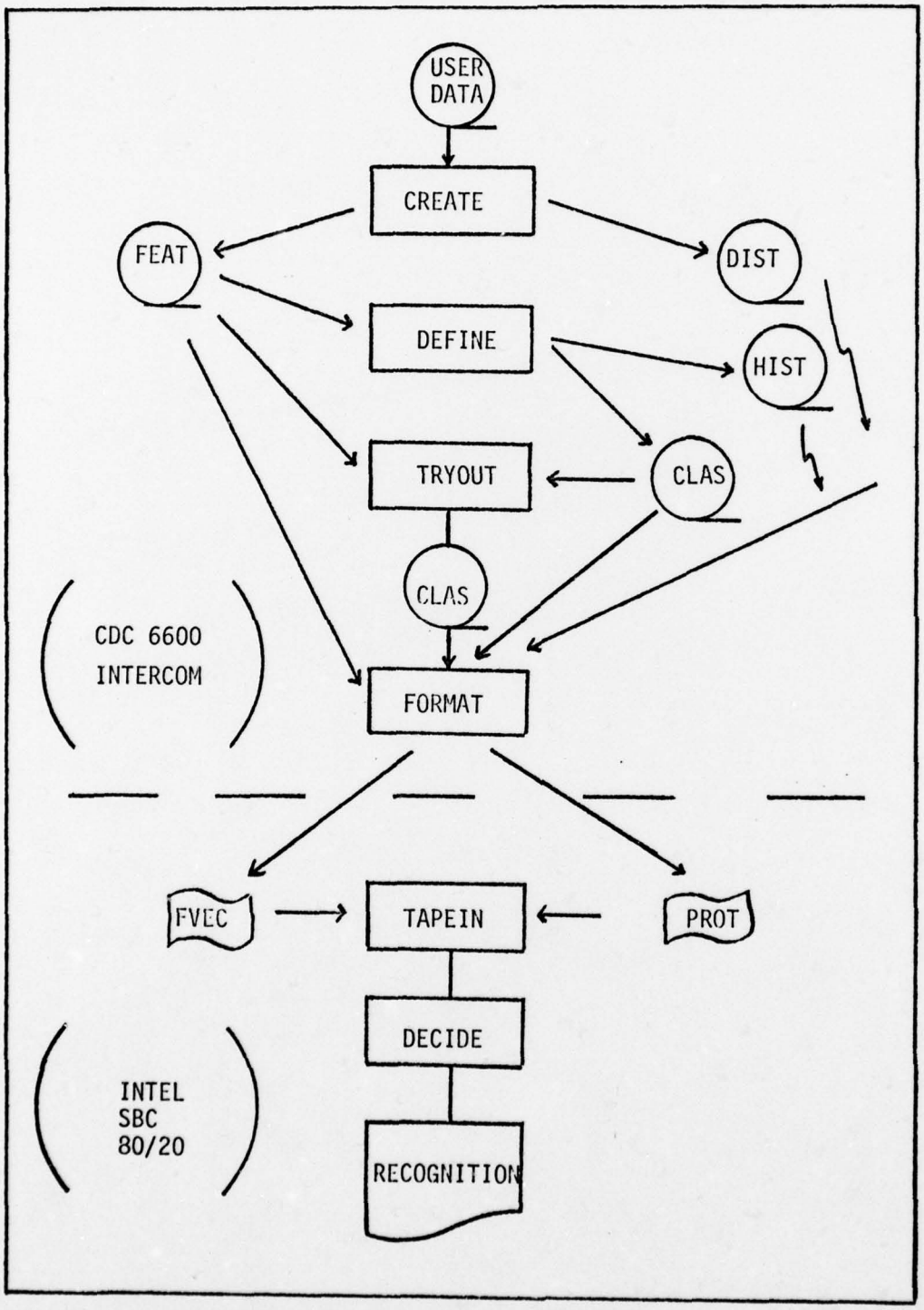


Fig. 24. BOX80 System Data Flow

TABLE III
MODULE AND ROUTINE NAMES

**SYSTEM MODULES:

```
PROGRAM CREATE(USER,FEAT,HISG,LOGF=64,INPUT=64,OUTPUT=64)
 SUBROUTINE DEFC(IEOJ,NW,JDX,NHX)
 SUBROUTINE SCAN(BUF,HISG,VECS,IEOJ,NW,NHX)
 SUBROUTINE COPY(BUF,FVEC,HISG,IEOJ,NW,JDX,NHX,VECS)
 SUBROUTINE GETFEA(IUSER,BUF,NW,FVEC,JD,IC,NVEC,IGC,IEOJ)

PROGRAM DEFINE(FEAT,OLDC,NEWC,HISC,LOGF=64,INPUT,OUTPUT)
 SUBROUTINE DEFD(IEOJ)
 SUBROUTINE ALLOC(KC,NE,HUSKR,IEOJ)
 SUBROUTINE NEXCLA(FEAT,CLAS,JDX,IEOJ)
 SUBROUTINE PHUSK(CLAS,JDX,IEOJ)
 SUBROUTINE KERPUT(CLAS,JDX,NEXC,IEOJ)
 SUBROUTINE CLASSX(FEAT,CLAS,HISC,JDX,NHX,IEOJ)
 SUBROUTINE CDEFI(FEAT,CLAS,HISC,JDX,NHX,IEOJ)
 SUBROUTINE FANDER(CLAS,JDX,FEAT,IEOJ)
 SUBROUTINE SETLIM(CLAS,FEAT,JDX,IEOJ)

PROGRAM TRYOUT(FEAT,OLDC,NEWC,LOGF=64,INPUT=64,OUTPUT=64)
 SUBROUTINE DEFT(IEOJ)
 SUBROUTINE MERIT(FT,DISI,DISN,CLAS,JDX)
 SUBROUTINE FIGM(FT,JDX,IEOJ)
 SUBROUTINE SUBSET(CLAS,JDX,ICX,IEOJ)
 SUBROUTINE EVAL(FEAT,CLAS,CMX,JDX,NAX,IC,IEOJ)
 SUBROUTINE DOCU(CMX,CLAS,JDX,NAX,IC)
 SUBROUTINE LOOK

PROGRAM FORMAT(FEAT,OLDC,HIST,BYTE,LOGF=64,INPUT,OUTPUT)
 SUBROUTINE DEFF(IEOJ)
 SUBROUTINE XCLAS(CLAS,BUF,HID,JDX,NBX,IEOJ)
 SUBROUTINE XHIST(HIST,BUF,HID,NHX,NBX,IEOJ)
 SUBROUTINE XFEAT(FEAT,BUF,HID,CLAS,JDX,NBX,IEOJ)
 SUBROUTINE FILBUF(HIST,BUF,NHX,J1,JN,NX,NY,ICON)
 SUBROUTINE XMIT
 SUBROUTINE PICT(BUF,HID,NX,NY)
 SUBROUTINE STRIP(BUF,NX,NY)
 SUBROUTINE NEXVEC(ISEQ,NVEC,NRB,IEOJ)
 SUBROUTINE NEXREC(IEOJ)

TAPEIN(EQU=NMOUT,GETCH,GETCM,CNUBN,ERROR)
 BYTEX(A=BYTE)

DECIDE(EQU=PROT,FVEC,JD,LB,IC)
 CLOOP(B=A.LEN,HL=A.STRING-LAST)
 OUTB(HL=A.VALUES,DE=A.OUTPUT,B=LEN)
```

**UTILITY ROUTINES:

```
 SUBROUTINE INITC(NENT,LIST,IADD)
 SUBROUTINE ADD(NAME,LIST,NENT,NPOS,IEOJ)
 SUBROUTINE DEL(NAME,LIST,NENT,NPOS,IEOJ)
 SUBROUTINE RIX(NAME,LIST,NENT,NPOS,IEOJ)
 SUBROUTINE INDEX(CLAS,JDX,ICX,IEOJ)
 FUNCTION KERGET(CLAS,JDX,NEXC,IEOJ)
 SUBROUTINE PRCLAS(CLAK,LIST,CLAS,JDX,ICX)
 SUBROUTINE LOADC(CLAS,JDX,ICX,IEOJ)
 SUBROUTINE OPENH(LABLH,IEOJ)
 SUBROUTINE OPENX(IMOD,IE,LABLC,LABLF,IEOJ)
 SUBROUTINE RFEAT(FEAT,JDX,INIT,IEOJ)
 SUBROUTINE RHIST(HISC,NHX,IEOJ)
 SUBROUTINE WRCLAS(CLAS,JDX,ICX,IS,LABLC)
 SUBROUTINE WRHIS(HISC,NEXC,ISYM,LABLC,NBUC,KTR,NI,NHX,JDX)

 SUBROUTINE STATX(LAS,FIN,FT,CAU,CSD,ISW)
 SUBROUTINE STATH(LAS,FIN,FT,FHIS,NI,FIRS,FLAS)
 SUBROUTINE XSCAL(FMIN,FMAX,VEC,NN,SCALE,IOP)
 GETCH(C=CHAR,A=CHAR)
 CI(A=CHAR)
 CNUBN(C=CHAR,A=BINARY)
 ERROR(ENTRY)
 GETCM(ENTRY)
 DIV(BC=DIVIDEND,D=DIVISOR,C=QUOTIENT,B=REM)
 HILO(DE=A.V1,HL=A.V2)
 BNBCD(DE=INPUT,HL=A.BUFFER)
 COUT(C=BYTE)
```

SUPPORT ROUTINES:

```
 FUNCTION ENER(VEC,N)
 SUBROUTINE MARK(IX,IY,IDX,IDY)
 SUBROUTINE PLOT3D(FCN2D,HID,IMAX,JMAX)
 SUBROUTINE PLX(X,Y,II)
 SUBROUTINE ILINE
 SUBROUTINE IASORT(IN,NN)
 SUBROUTINE FDSORT(FIN,ITAG,FOUT,NN)
 ERROR(ENTRY)
 GETCM(ENTRY)
```

TABLE IV (1/3)
MODULE AND ROUTINE DEFINITIONS

1. CREATE - Generates FEAT file from user data
   - DEFC - Initializes CREATE module
   - SCAN - Produces "first-pass" statistics on features
   - COPY - Generates FEAT file records
   - GETFEA - Reads user data file
   - PRHIST - Prints statistics and histograms
2. DEFINE - Generates CLAS file from FEAT records
   - DEFD - Initializes DEFINE module
   - ALLOC - Allocates memory to module buffers
   - NEXCLA - Controls selection of class to be processed
   - KERPUT - Updates class husk list
   - CLASSX - Controls processing of class data
   - CDEFI - Updates prototype definitions and histograms
   - FANDER - Produces feature vectors as husk members
   - SHUCK - Identifies feature vectors as husk members
   - SETUM - Inserts feature boundaries into CLAS file
3. TRYOUT - Produces error rates and feature subsets
   - DEFT - Initializes TRYOUT module
   - MERIT - Computes figure of merit for each feature
   - FIGM - Presents and accepts feature merit ranking
   - SUBSET - Tags dimensions for elimination
   - EVAL - Performs trial recognition
   - DOCU - Outputs error rate and confusion matrix
   - LOCK - Establishes subspace error rates
4. FORMAT - Produces microprocessor data and displays
   - DEFF - Initializes FORMAT module
   - XCLAS - Controls processing CLAS file
   - XHIST - Controls processing HIST AND DIST file
   - XFEAT - Controls processing FEAT file
   - FILBUF - Loads buffer with PICT and STRIP input
   - XMIT - Sends values to hexadecimal format routine
   - NEXREC - Inputs user selection of data class
   - NEXVEC - Inputs user selection of vector

TABLE IV (2/3)
MODULE AND ROUTINE DEFINITIONS

5. TAPEIN - Decodes and loads cassette tape into SBC 80/20 ROM
   - BYTEX - Reads a pair of hexadecimal characters
6. DECIDE - Microprocessor classifier module
   - CLOOP - Outputs a string of characters
   - OUTB - Outputs a buffer of binary values as characters
7. UTILITIES - [General Purpose System Routines]
   - INITC - Initializes CLAS file index chain
   - ADD - Adds entry to CLAS file index chain
   - DEL - Deletes entry from CLAS file index chain
   - RIX - Reads CLAS file index chain
   - INDEX - Builds CLAS file table index; scales file
   - KERGET - Accesses CLAS file husk list
   - PRCLAS - Prints CLAS file
   - LOADC - Loads CLAS file buffer
   - OPENH - Opens HIST AND DIST files
   - OPENX - Opens FEAT and/or CLAS files
   - RFEAT - Reads FEAT file record
   - RHIST - Reads HIST file record
   - WRCLAS - Writes CLAS file record
   - WRHIS - Writes HIST file record
   - STATH - Updates histogram
   - STATX - Updates statistics
   - XSCAL - Scales FEAT and CLAS vectors
   - GETCH - Reads a character (SBC 80/20)
   - CI - Input from RS232 port (SBC 80/20)
   - CNVBN - Converts to binary (SBC 80/20)
   - DIV - Divides 16 bits by 8 bits (Interp 80)
   - HILO - Compares 16 bit values (SBC 80/20)
   - BNBCD - Binary to BCD conversion (INTEL User Library)
   - COUT - Character output routine (SBC 80/20)

TABLE IV (3/3)
MODULE AND ROUTINE DEFINITIONS

SUPPORT - (Specialized Support Routines)

ENER	- Computes 'energy' and string of values
MARK	- Draws tic mark on TEKTRONIX screen
PLOT3D	- Hidden line routine draws 3D surfaces
PLX	- Emulates CALCOMP plot routine
ILINE	- Generates Intel hexadecimal byte format
IASORT	- Integer ascending sort
FDSORT	- Floating point descending sort
ERROR	- Generates error prompt (SBC 80/20)
GETCM	- Gest next user command (SBC 80/20)
ERR	- Generates error prompt

System Subroutines

In this section standard supporting techniques for data manipulation are discussed. Additionally abstracts of utility and support routines are presented.

Module Initialization. Each system module is initialized by a subroutine which establishes standard file names, and allocates memory from a single work area to the file buffers and tables required for processing. Record block sizes within each system file are set by the user at system initialization. These two techniques simplify transport of the prototype generation segment from one FORTRAN capable system to another. They allow adjustments for memory and on-line storage variations in different systems. Record block sizing algorithms establish pointers to starting locations of module buffers by iteratively adjusting data parameters. These are then output for user approval of buffer size adjustments.

File Processing. In order to design efficient structures for feature vector and prototype data, usage and access patterns were analyzed. A file structure was selected over the use of incore buffers so as to allow greater data volume. Implementation using separate modules was selected in order to enhance transportability. The requirement to generate prototypes via an interactive, time sharing process raised questions about both memory and execution time limitations. Execution of separate

modules is consistent with use of minimum amounts of core and time to complete a given function. A standard file structure was established for data communication between modules. This structure consists of four files which are defined in the next section.

Requirements to access each file were analyzed in the process of defining structure. The feature vector data has greatest potential volume and least need for non-sequential access. Conformance to ANSI FORTRAN specifications dictated a sequential access method but allowed a BACKSPACE operation. Thus a disk or tape based sequential file was selected for this data. However, prototype data is accessed frequently in iterative processing, and is not necessarily only used sequentially. Again conforming to ANSI FORTRAN capabilities dictated use of a sequential file. However, since its volume is limited, a single record approach was chosen. Use of an embedded index to the data vectors associated with each prototype supported efficient use of in-core storage of this record. This technique also supports revision to a multi-record random access file structure in environments, such as with minicomputer hosts, in which there is extremely limited central memory. Finally, histogram data appeared to be too voluminous for incore storage, and too infrequently used for a multi-file solution to the restriction posed by the ANSI sequential file standard. Therefore, two files

of the same format were designed. One, with a single data record, contains universe distributions. The other, containing one record for each data class, records distributions of data within each data class. These four files are labeled DIST (universal distributions), HIST (class distributions), CLAS (prototypes) and FEAT (feature vectors).

Utility Routines. There are three types of subroutines within the BOX80 system. In the first group are routines uniquely specialized to support primary modules. These are covered in the next chapter. General purpose utility routines are synopsized below. Table D-V gives calling parameters and their definitions. Special purpose support routines having general usefulness are discussed in the next paragraph.

(1) ADD. One of four routines which access the index to the prototype data file, this routine inserts a new entry to that index. The index is described in the next section. It contains two chains of entries. The entries in one chain correspond to column vectors in the CLAS file data record. The entries in the other chain correspond to unused column vector positions. This ADD routine follows the 'used' entry chain to the appropriate position and relinks an index entry to hat position from the top of the 'used' chain.

(2) DEL. This routine deletes an entry from the index to the prototype data file. Deletion is effected by relinking

around the indicated index entry and adding the newly freed entry to the unused chain. This routine does not clear the associated column vector; it is therefore uncoupled from its referenced data area. This eases data structure modifications.

(3) INITC. This routine sets constant parameter into the prototype data file index during file initialization. See Figure 25 for a sketch of these initialization entries.

(4) RIX. This routine reads the index to the prototype file and extracts the entry number of the named vector. The appropriate index entry is found via a sequential search of the 'used' entry chain.

(5) INDEX. In order to speed retrieval of the address of named prototype data, this routine builds a table of 51 three position entries. Each position records the address of a prototype vector. Entry 51 records the address of a pair of data limits vectors. At option, this routine controls scaling of prototype data into a specified bit range.

(6) KERGET. A set of vectors within the prototype file records identifiers of feature vectors which have been assigned to the husk of each class. This routine obtains the identifier for the "next" feature vector assigned to the husk of a given class.

(7) PRCLAS. This utility routine prints the three data types stored within the prototype file. The file index, the set of husk vectors for each class, and the prototype definition for the class are printed.

(8) LOADC. This routine loads the prototype data file into the proper program buffer.

(9) OPENH. This routine reads the header record from HIST and DIST files setting the x-dimension memory parameter associated with the data records of the file.

(10) OPENX. This routine reads header records from FEAT and CLAS files. The parameters set include the y-dimension memory variable corresponding to number of data columns within the CLAS file. These open functions are coupled so that label tests can be made in one place. At input options, the FEAT file or the CLAS file open can be bypassed. This is necessary when the CLAS file is either used alone, or is to be initialized or extended in size.

(11) RHIST. This utility reads records from either DIST or HIST files. A sequential search is made for the requested record, and no backspace or rewind option is provided. Records containing histogram pairs are flagged. An error indicator is set if a missing record is requested and an end job flag is set when an illegal record is requested.

(12) RFEAT. This utility routine handles input from the FEAT file. A rewind option and a backspace option support reprocessing the entire file or the current record block. A sequential search is made for the requested record. Missing records cause a fatal error flag to be set. Each block is

checked for the last block flag; a pointer to the last vector of the block is updated when this block is read.

(13) WRCLAS. This subroutine writes the CLAS file from memory onto its file. In this process it assembles and outputs the CLAS file header.

(14) WRHIS. This subroutine writes the HIST and DIST file records. It assembles and outputs the file header record, and provides a printout of statistics and distribution values if requested.

(15) STATH. This routine generates a histogram from the stream of values input on successive calls. The current histogram is always output. At receipt of a last-call indicator a mode and the percent of all values associated with this mode are calculated.

(16) STATX. First and second order moments, minimum and maximum values are computed from a stream of input values. Temporary values are initialized at first input which is signalled to the routine by a zeroed work area parameter. The current minimum and maximum are always output. A last call indicator triggers generation of mean and variance values. At option either the standard deviation from the computed mean, or an average deviation from a zero mean is computed. The latter deviation is used for definition of assymmetric class boundaries.

(17) XSCAL. Components of vectors input to this routine are scaled to a specified range of values. Either of two scaling

algorithms is selectable for use with prototypes or with class diagonal covariance matrices. (See Chapter IV.)

Support Routines. The subroutines described below each support unique functions within the BOX80 system. However, since these routines have a conceptually general utility they are discussed as a group here. Table D-VI gives calling parameters and their definitions.

(1) MARK. This routine generates a tic mark, of specified direction and length, on the TEKRONIX screen.

(2) ENER. This routine computes the magnitude squared, or sum of the squares of the component values of any vector.

(3) PLOT3D. This subroutine executes a hidden line analysis and perspective transformation in order to produce a three-dimensional plot of a two-dimensional array containing z-axis values. The subroutine listing contains added comments and a source reference.

(4) PLX. This routine simulates the CALCOMP routine PLOT insofar as necessary to translate the capability of PLOT3D from CALCOMP to TEKTRONIX output.

(5) ILINE. This routine reformats a string of 16 integer values into a paper tape data format used on various microprocessor systems. This line format is presented in Table D-VIII. A process switch controls operation of this routine. It allows generation of initial line address characters and output of special

end of record line format. Integer to hexadecimal encoding uses only ANSI standard FORTRAN operations.

(6) IASORT. This routine sorts an input array of N integers into ascending order. The input data sequence is lost.

(7) FDSORT. This routine sorts a pair of input arrays into descending order based on the real value of members of one array. It preserves the input data value sequence.

(8) DIV. This 8080 routine divides a 16-bit dividend by an 8-bit divisor producing an 8-bit quotient and an 8-bit remainder (Ref 15:18).

(9) BNBCD. This 8080 routine converts a 16-bit, two byte integer into a string of 5 ASCII character codes (Ref 19:18).

(10) HILO. This 8080 routine compares two 16-bit unsigned integers and sets the 8080 carry condition indicator to show a less than or equal condition (Ref 18:B-26).

(11) COUT. This routine sends a single byte to the RS232 port of an 8080 system if the port is ready to write.

(12) CNVBN. This 8080 routine converts the BCD represented hexadecimal characters to their integer values (Ref 9:8-23).

(13) GETCH. This 8080 routine reads ASCII valued characters and strips the parity bit (Ref 19:B-20).

User Input Routine. To make the CREATE module as general as possible a user supplied routine is referenced to read the user file of feature data. The specifications for this module are described below.

GETFEA. This routine reads the file of feature data supplied by the user. Its calling parameters are defined in Table D-VIII. On the first entry to this routine, the user data file is rewound. On the first and all successive entries a feature vector is returned. Additionally a feature vector identifier (number of the vector within its class), a class identifier, and an error flag are returned on every call. If the vector returned is not the last of its class, this flag is set to zero. If this vector is the last of its class, this flag is set to -1. If this fector is the last of the file, this flag is set to +1. Once the last vector of the user file has been output the routine must reset all internal flags to allow for rewind on the next call. Input to the routine includes a file name and buffer location with size as well as an option switch with nine settings for use as desired.

Data Structures

In this section descriptions are presented for each of the data structures used within the BOX80 system. Separate paragraphs below discuss the design of each file and define its format. There are six system files. Associated with one file is an index structure which is described separately. The system names for these files are stated with brief definitions below.

(1) FEAT. This file contains feature vectors ordered by class.

(2) CLAS. This file contains class prototype definitions and an imbedded index (referenced as LIST) to class definitions.

(3) DIST. This file contains component statistics and distributions on the population of data vectors input to the CREATE module.

(4) HIST. This file contains statistics and distributions on the data within each class processed by the DEFINE module.

(5) PROT. This file contains prototype definitions in a format suitable for use by the DECIDE module.

(6) FVEC. This file contains feature vectors in a format suitable for input to the DECIDE module.

Features Data File (FEAT). This file is ordered by pattern class and consists of multi-block records with one record per pattern class. Each data block has the same format and has a fixed size. This size is fixed at file creation, as earlier stated, to give the user control over use of his available memory resource. The first record of the file is a single header block. The last record of the file is a single trailer block. Formats for these three block types are given in Table A-I. The file is produced by the CREATE module which reformats input data from a user file and adds several tag values to each data vector.

The FEAT file structure is designed to allow variably sized data sets to be retrieved efficiently. Each vector within a block is tagged with its own identifier and the identifier of its class for ease in documenting trial error rates, as well as

for convenience in manually referencing file content. The header record was required to allow input of data into a variably sized buffer when the file is read. Its critical parameter gives the x-dimension of this input data buffer. The trailer record stores minimum and maximum component values for later use in scaling feature vector components to a byte-sized value range.

Class Definitions File CLAS). This file consists of two records. The ten-word header record identifies the file and establishes the size of the prototype data record. This size is set by the user during operation of the DEFINE module. The data record contains a set of vectors and a file index known as LIST. This index will be discussed in another paragraph. Data vectors are of three types. Each class is defined by a subset of these vectors containing from two to nine elements. Vector types include a class prototype or mean vector, a class boundary or deviation vector, and a class husk vector. Prototype and deviation vectors are directly used in the classification algorithm. The husk vector is used in the identification of candidate feature vectors for the formation of mean and deviation values. Deviation vectors represent an uncorrelated covariance of feature vector components within the class. A processing option allows these vectors to represent positive and negative zero-based deviations of class feature vectors from the mean vector. In either case,

the classification algorithm operates upon these deviation vectors as diagonalized matrices. Thus the term vector is used at this point only for parallelism and simplicity since a vector is indeed a linear list of components. Formats for the file header and for these vectors are given in Tables A-II and A-III, and in Figs. 25 to 27. (Note: The structure of the data buffer for the CLAS file data record is intricate. The content of this buffer is varied. The referenced figure and tables must be examined as a set in order to understand this structure and content.)

Column vectors within the CLAS file data record have JD dimensions and have three tags. These tags extend column size to JDX. Tag three, for all vector types, represents the class id. Tag two carries a code indicating vector type. Tag one stores two types of value: for class mean and deviation vectors it contains the least and greatest component values in the class for component scaling in the cluster plot process; in class husk vectors it contains the number of the next open vector component for use in husk manipulations. Tags one and two are used as identifiers for their respective vectors in character printouts of this data record.

Class Definition File Index (LIST). This structure is an array with two items per entry, having two more entries than there are columns for data vectors within the CLAS file data record. These entries provide an index to the data record. The

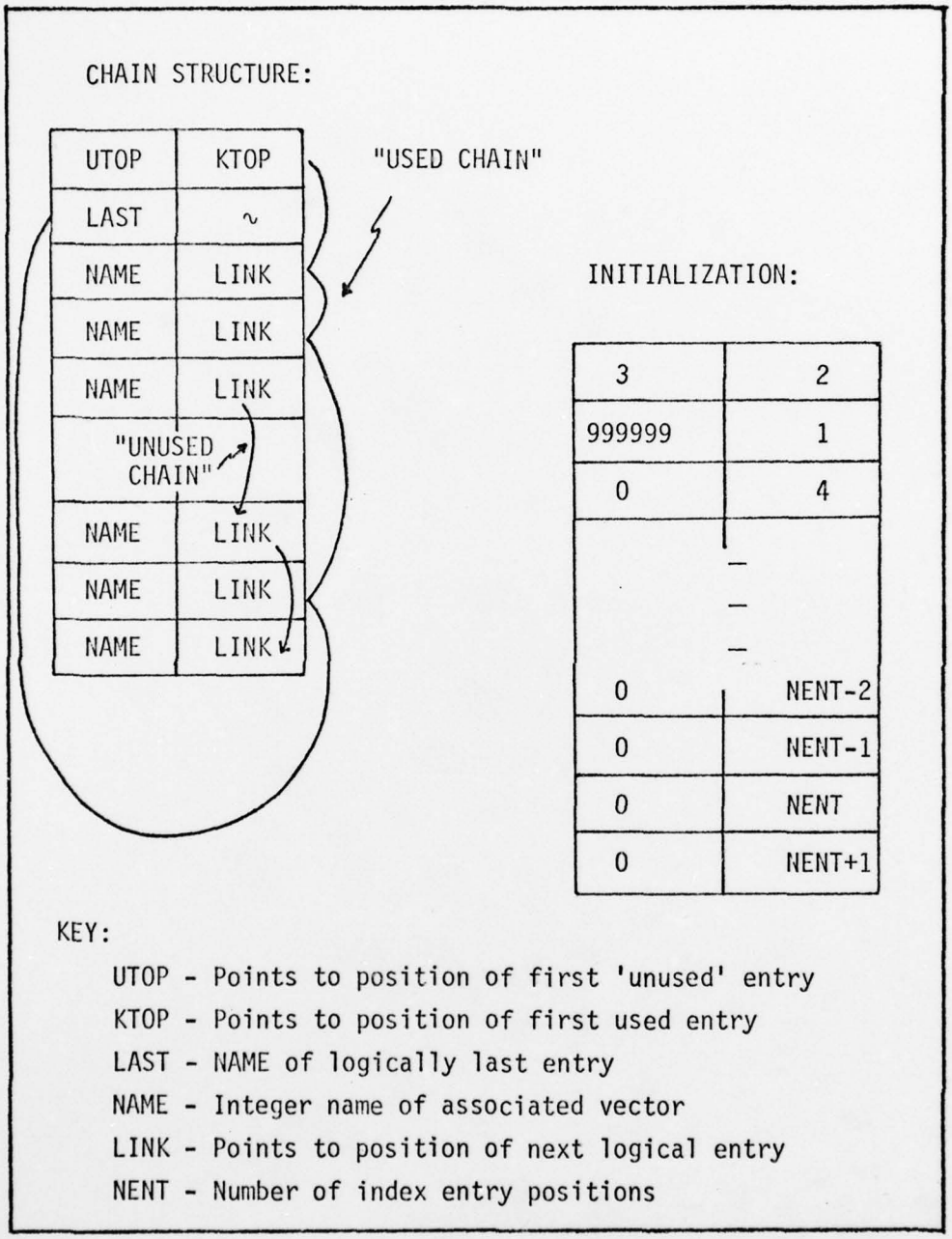


Fig. 25. CLAS File Index Structure

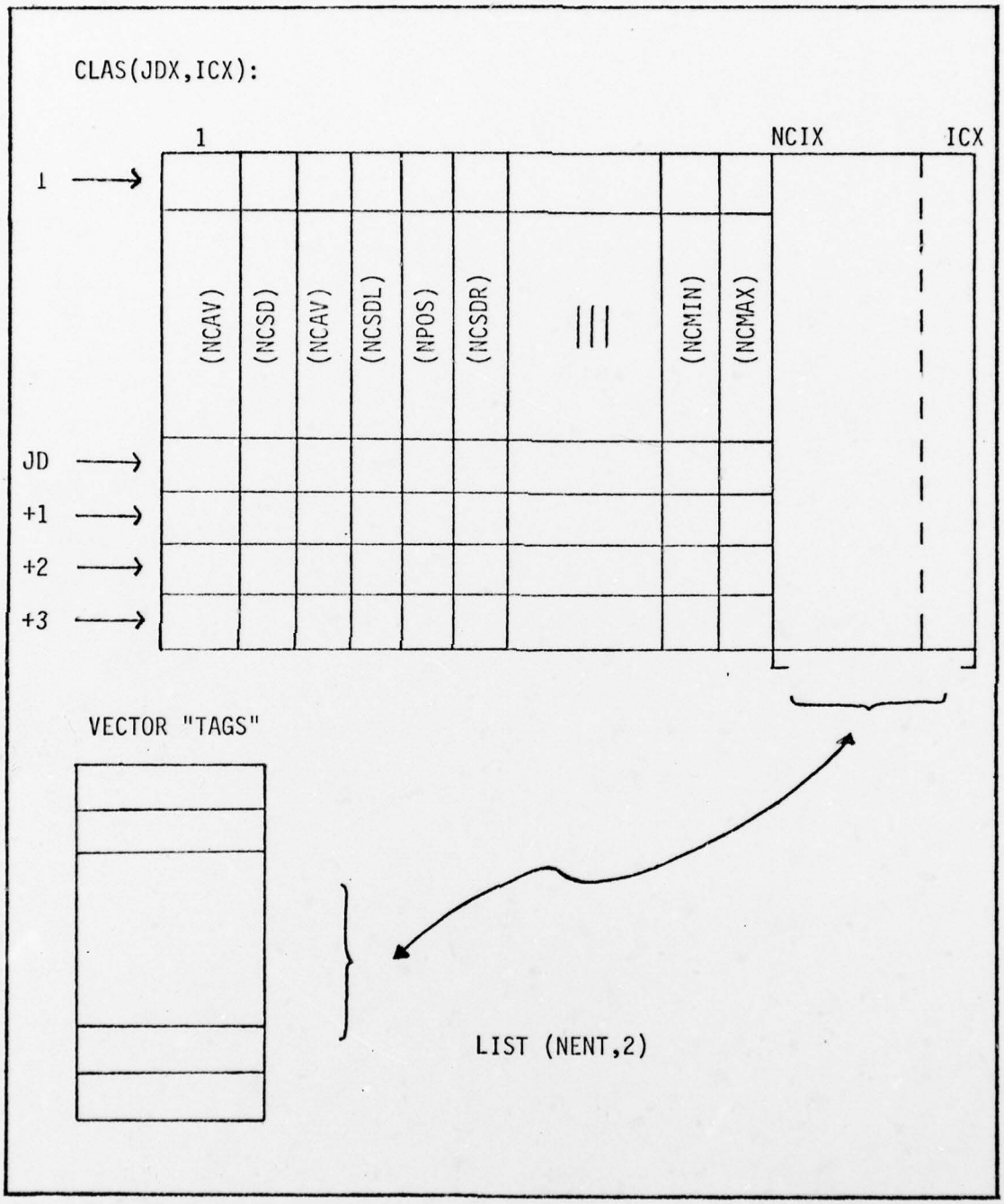


Fig. 26. Diagram of CLAS File Structure

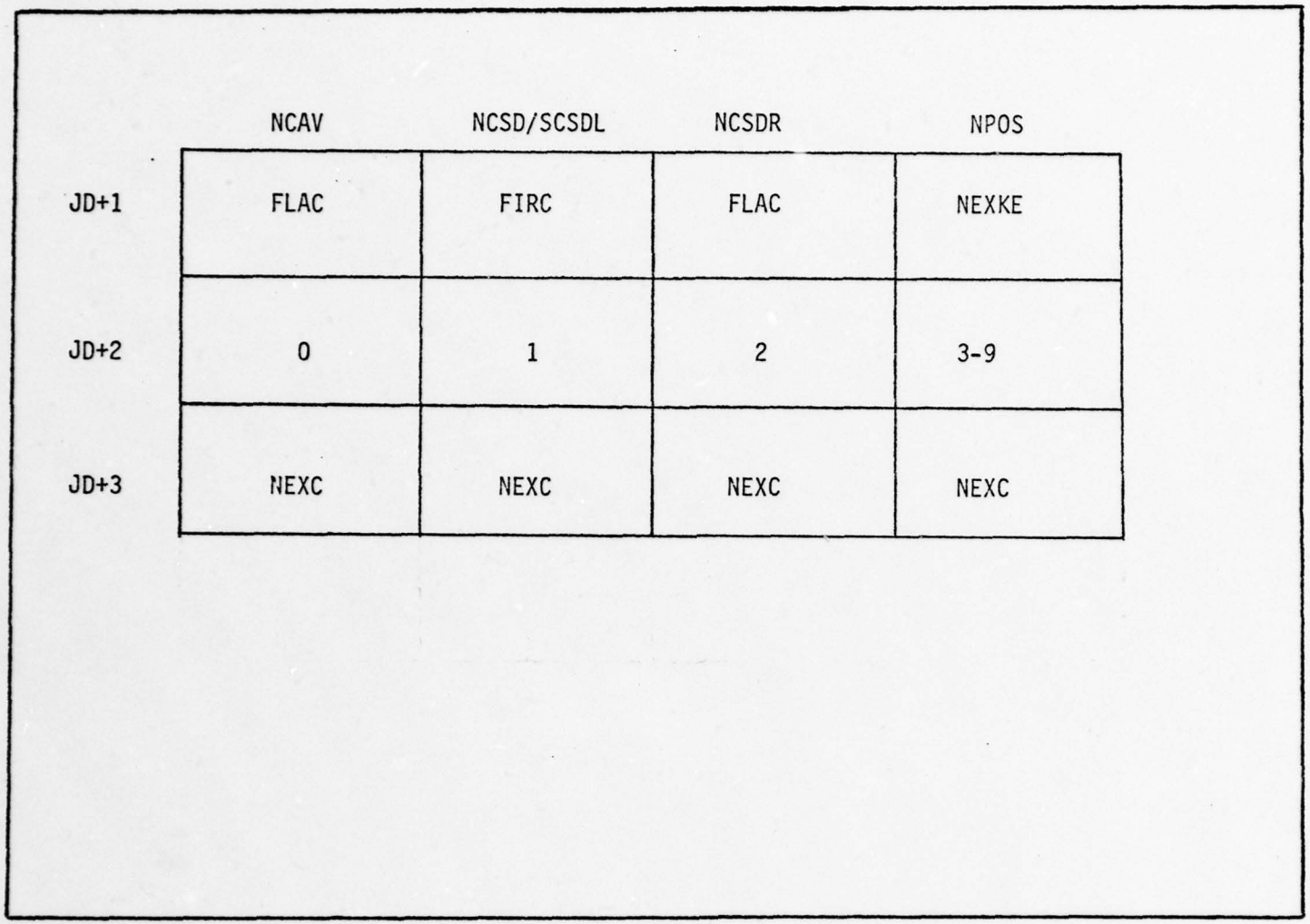


Fig. 27. CLAS File Data Record - Vector Tags

first item of each entry contains a code for the name of a given vector in the data record. The second item of the entry contains a pointer to the list entry position which contains that entry which logically follows the present entry according to the value of its name. There is obviously one entry in this index for each column vector in the CLAS file data record. The two extra entries in this index are described by Fig 25. The first of these entries is always the first entry in the array. Its first item points to the first unused entry of the array. Its second item points to the first used entry in the array. The second entry in the array is a dummy entry whose name indicates logical 'last'. The pointer item in this entry is arbitrary since the index entry chain is not circular. These entries are used by the index service routines to maintain the logical chain of name items. Because of these two entries, whose physical positions are fixed, the name-item in each functional entry in the array refers to a column vector in the data record whose column number is two less than the list entry position of that name-item.

This technique of indexing the column vectors within the CLAS file data record was chosen for two reasons. It allows non-sequential generation of each of the vectors within the prototype set for a given class without requiring the reservation of a fixed amount of space for the vector set for each class. Moreover, it supports convenient revision of the memory allocation

to the CLAS file data record by allowing record extension under program control as well as straight-forward modification of the in-core data structure. This latter fact adds to transportability of the prototype generation segment of the BOX80 system.

The name items used in this index array are structured numbers. Their form is given by the expression

NAME = I * 1000 + K

in which I is the class identifier and K is one of the following numbers:

K = 100 for Mean vectors

K = 201 for Negative deviation vectors

K = 202 for Positive deviation vectors

K - 30n for Husk vectors, n

Distribution Data File (DIST). This file consists of two records. The first is a header record. The second is a data record which contains statistics and histograms for each feature component's values as they occur within the entire population represented by the FEAT file. The DIST file is generated by the CREATE module. Table A-IV describes the record format and defines the data items within this file. The FORMAT module processes this data. It provides graphic displays of histograms for analytical use.

Histogram Data File (HIST). This file consists of records generated by the DEFINE module. HIST file data records are

processed by the FORMAT module to produce graphic displays for analytical use. The header record for this file is identical in format to that of the DIST file. The data records of this file are fixed in size, but have a variable format. Size is fixed to the space allocated by the DEFINE module. The record format varies in order to allow storage of histograms output by the prototype revision process when assymetric classes are defined. In the symmetric format the histogram data area for each feature dimension contains one set of eight statistics and one set of interval counters which store the histogram. In the asymmetric format, two sets of statistics, and two sets of histogram counters are maintained. The added pair of sets appears within the array starting at the position indicated by the variable KTR. The variable NI gives the number of histogram intervals maintained in the asymmetric case. The variable NBUC gives the number of intervals for the symmetric case.

Prototype Data File (PROT). This file consists of a set of class defining data records which are encoded in a superimposed data format. The latter format facilitates transfer of the prototype definitions from the prototype generation segment of the BOX80 system to the classifier segment of the system. It is described in Table D-VII. This format is a data standard (Ref 17) on the INTEL SBC 80/20 microprocessor used for execution of the classifier segment. With minor variations, it is also used on Motorola 6800 systems (Ref 26). The format for the actual

prototype data values is given in Table A-V. Each class definition in this prototype format consists of a string of values which define the region of feature space assigned to the class. These values include the prototype mean and positive and negative deviation values for each feature dimension. Each such string of values is preceded by a class identifier.

Feature Vector File (FVEC). This file consists of feature vector components and vector identifiers. These values are ordered for processing by the classifier segment of the BOX80 system. Data in this file is encoded in the hexadecimal format presented in Table D-VII. The structure and content of a feature vector record in this file is presented in Table A-VI. This file is produced by the FORMAT module of the prototype generation segment for use in test processing. It demonstrates the feature vector structure processed by the classifier segment.

Interpreter Segment

This segment consists of four modules. These are CREATE, DEFINE, TRYOUT, and FORMAT. Separate subsections define the design of each of these modules.

CREATE. This module builds a file (FEAT) of feature vectors in BOX80 system format for later system processing. A file (DIST) containing statistics and histograms for all vector components processed may be output. Various vector transforms are possible. Output of a statistics report as well as certain

execution trace information can be provided on the LOGF file. CREATE functions are described below with the subroutine which implements them. Fig 28 presents a data flow chart for CREATE. A structure diagram is given in Fig 29.

CREATE is composed of four major functional routines. These are DEFC, SCAN, COPY, and GETFEA. The first three are part of the BOX80 system; the latter is intended to be a user supplied routine. The sequence of subroutine calls within this module and the input/output parameters for specialized CREATE subroutines are presented in Tables V and B-I. Functional abstracts of these routines follow.

(1) DEFC performs module initialization functions: file names are set, and user options are requested, error checked and set. Memory is allocated according to an algorithm which sizes histograms and FEAT file record blocks. Space is allocated as requested to a user buffer for input of user data through routine GETFEA.

(2) SCAN accumulates statistics on feature vector components obtained from the user data file. A summary printout is provided at user option.

(3) COPY processes the user data input, and builds the output FEAT file. Vector transforms are exercised at user option prior to output of FEAT file records. Statistics and histograms are generated for these output feature vectors.

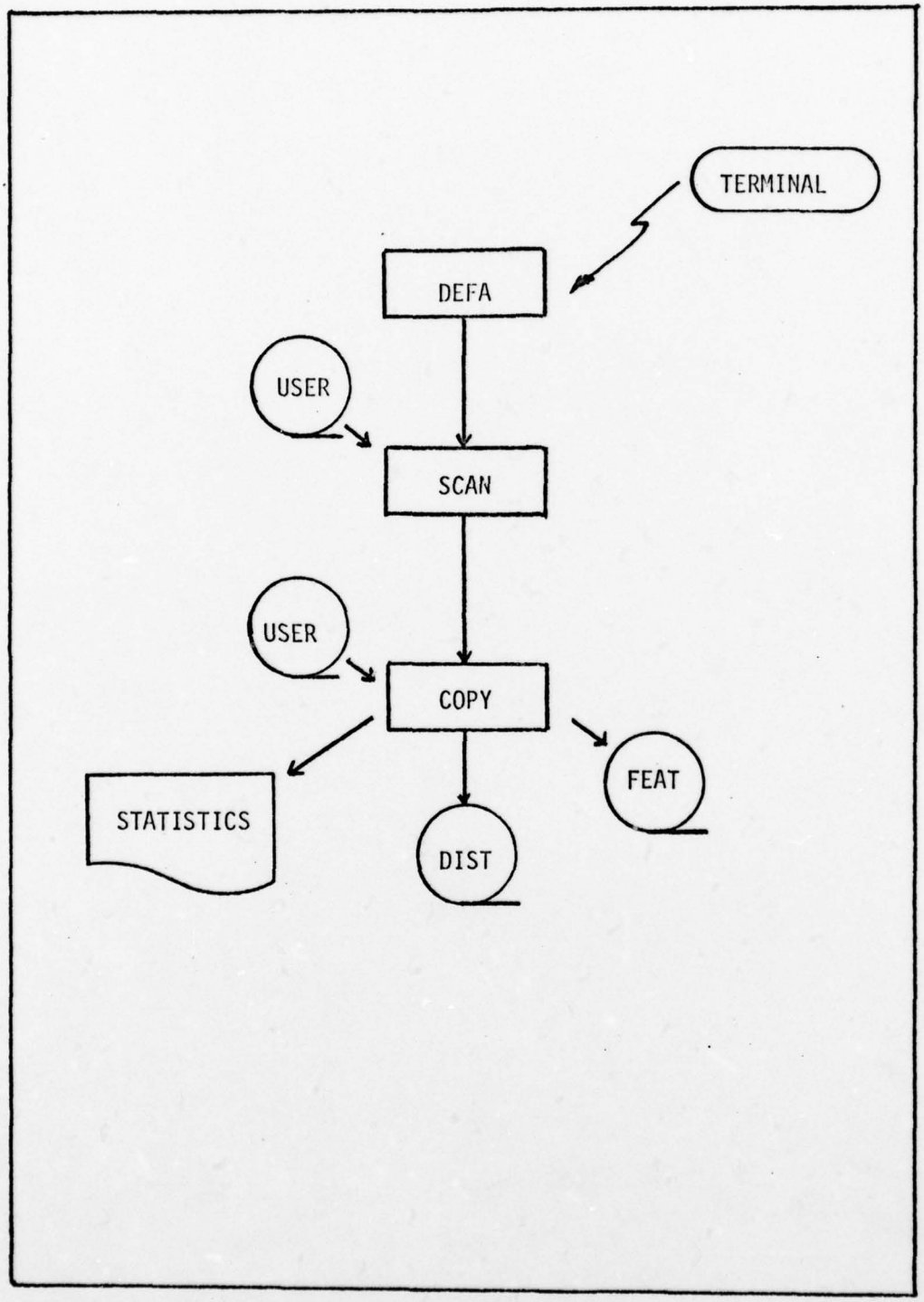


Fig. 28. CREATE Data Flow

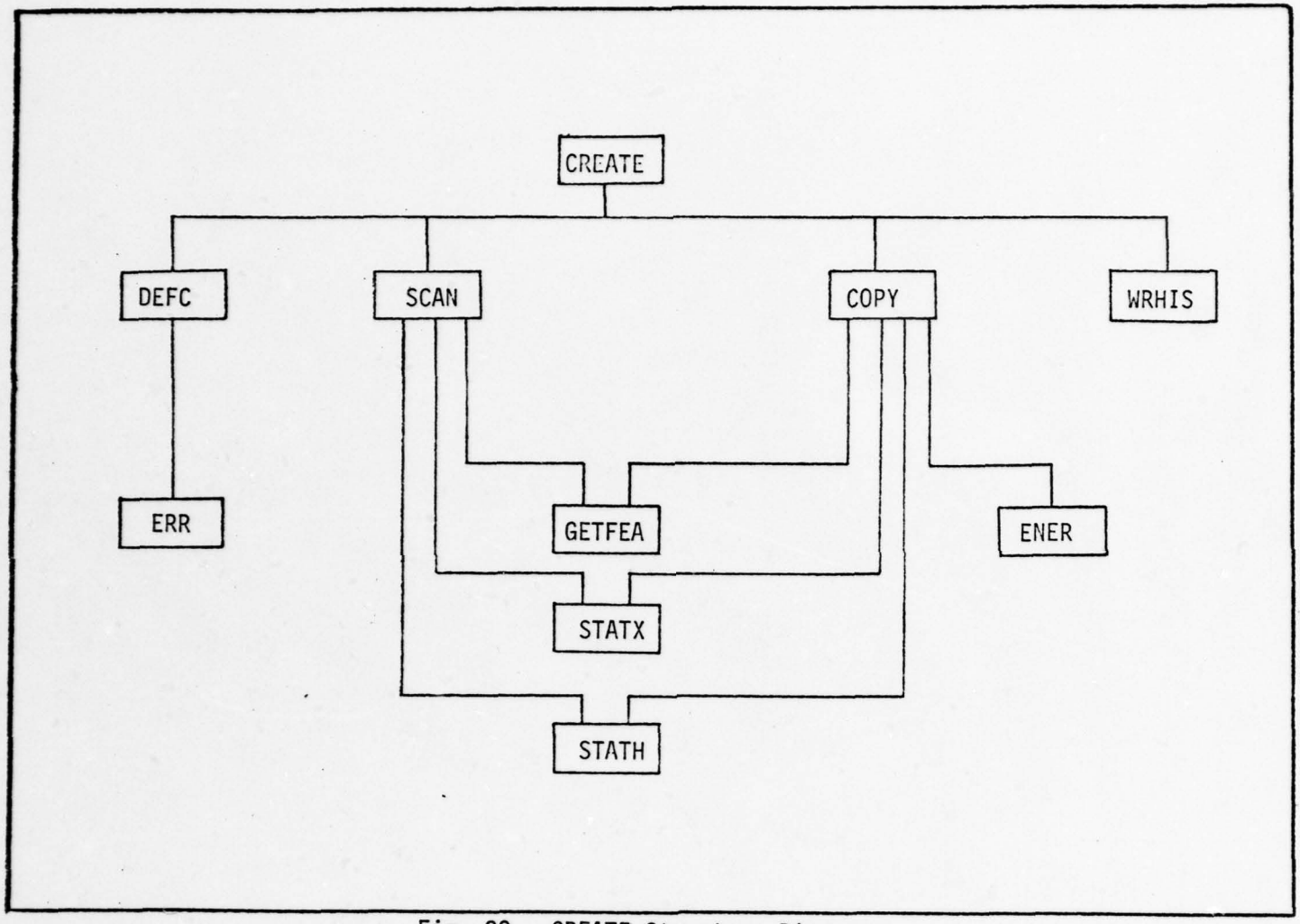


Fig. 29. CREATE Structure Diagram

TABLE V

Sequence of Calls CREATE Process

Routine	Description
CREATE	Generates FEAT file from user data
DEFC	Requests and sets module parameters
ERR	Echoes error prompt to terminal
SCAN	Collects statistics on users data set
*GETFEA	Reads feature vector from user file
STATX	Updates statistics
COPY	Copies user data set into FEAT file form
*GETFEA	Reads feature vector from user file
ENER	Computes the energy in a set of values
STATX	Updates statistics
STATH	Collects multivariate histogram
WRHIS	Writes HIST file record

*Underlined routines are unique to CREATE

(4) GETFEA is a file read routine supplied by the user. Three sample GETFEA routines are listed in Appendix E. Table D-VIII summarizes specifications for this user input module in terms of its input/output parameters.

CREATE processing consists of initialization followed by a one (or two) pass process through the user data-file. Vector transforms which can be selected for the output feature vectors establish standard value ranges for component dimensions which make comparisons of interclass histograms convenient by effecting a linear shift of component values. The energy normalization, unitzation and vector shift transforms affect feature vector magnitudes but preserve relative angles. The squaring transform varies both vector magnitudes and angles in order to extract as much precision from vector components as possible. Control options are given in Table B-I. Outputs to the LOGF file include a trace of subroutine exits, and a dump of input data records as well as printouts of data base statistics and histograms. Appendix K contains a sample of selected LOGF output. Table C-I briefly summarizes the each possible LOGF output.

DEFINE. This module generates and revises the CLAS file. It can be used to shuck sets of feature vectors so as to isolate the kernel of patterns most acceptable for use in prototype generation. It can be used to enlarge the structure of a CLAS file so as to allow for generation of sink-prototypes. Prototypes

can be generated singly or as an entire set. This process defines hyperrectangular regions in feature space which may be either symmetrical or asymmetrical about the mean of a class of feature vectors. The primary output of the module is a CLAS file. Secondarily, a HIST file may be output. This will contain histogram records for each class of feature vectors processed. The process of shucking sets of feature vectors is supported by a graphic display of vector component overlap. In addition to this support, a control structure and dummy calls are provided at points appropriate for interactive and automatic selection of husk feature vectors. Each function of DEFINE is described below with the subroutine by which it is implemented. Fig 30 presents a data flow chart for DEFINE. Fig 31 presents a structure diagram.

DEFINE is composed of three major subroutines and many supporting utility routines. These major subroutines are NEXCLA, CLASSX, and CDEFI. The supporting routines unique to DEFINE are DEFD, ALLOC, PHUSK, KERPUT, KERGET, FANDER and SETLIM. The sequence of subroutines called in a simple execution of this module is given in Table VI. The parameters for subroutines unique to this module are defined in Table D-II. An abstract of each of these routines is given below. Routines appear in execution sequence.

(1) DEFD initializes DEFINE. Table B-II defines input control options provided by this routine. Program parameters

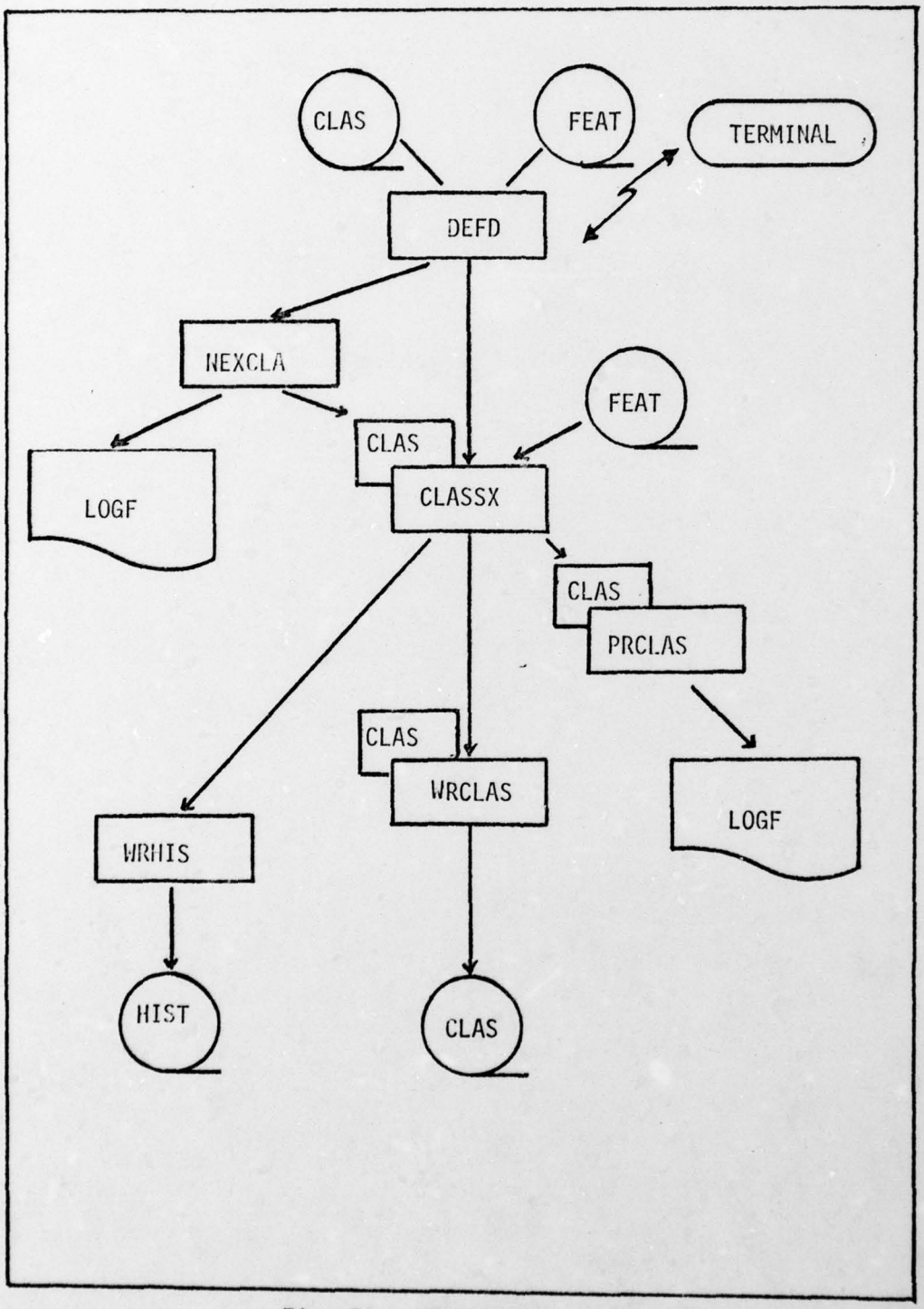


Fig. 30. DEFINE Data Flow

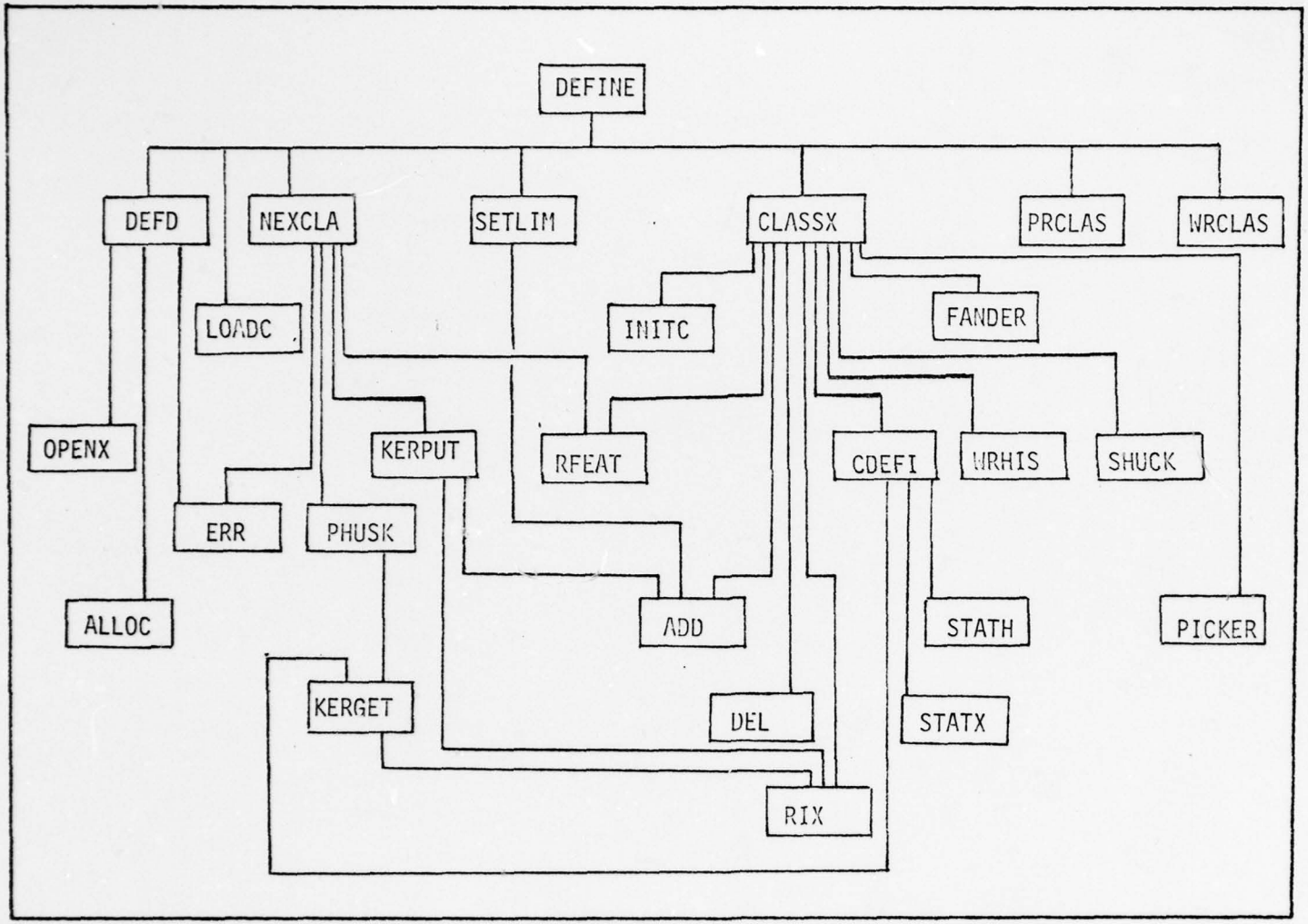


Fig. 31. DEFINE Structure Diagram

TABLE VI

Sequence of Calls in DEFINE Process

Routine	Description
DEFINE	Generates CLAS file from FEAT records
DEFD	Defines module parameters
OPENX	Opens FEAT and CLAS files
ALLOC	Allocates memory to initial CLAS file
ERR	Echoes error prompt to terminal
LOADC	Loads existing CLAS file
NEXCLA	Sets pointer to new class and obtains controls
RFEAT	Reads FEAT record
ERR	Echoes error prompt to terminal
PHUSK	Prints prototype husk list
KERGET	Gets entry from husk
RIX	Finds husk list entry in CLAS file
KERPUT	Puts entry into husk list
RIX	Finds husk list entry in CLAS file index
ADD	Adds a husk list entry to CLAS file
DEL	Deletes a husk list entry from CLAS file
CLASSX	Generates a prototype for this class of FEAT data
INITC	Initializes index to CLAS file entries
RIX	Finds prototype entries in CLAS file index
ADD	Adds prototype entries into CLAS file index
DEL	Deletes prototype entries in CLAS file index
RFEAT	Reads FEAT record for this class
FANDER	Produces cluster plot of feature vectors
CDEFI	Updates prototype component definitions
KERGET	Gets entry from husk
RIX	Finds husk for this class
STATX	Updates statistics for this class
STATH	Updates histograms for this class
WRHIS	Writes HIST file record
PRCLAS	Prints CLAS file record

TABLE VI (Continued)

Sequence of Calls in DEFINE Process

SETLIM	Inserts feature bounds into CLAS file
ADD	Adds entry to CLASS file index
RFEAT	Reads FEAT record
WRCLAS	Writes CLAS file to disk

*Underlined routines are unique to DEFINE

initialized by DEFD include the block of file names referenced by all input/output statements and the set of memory parameters which establishes the size and structure of the CLAS file. Comments in the listing of this routine provided in Appendix clearly define these parameters. DEFD controls allocation of memory to the CLAS file through OPENX (for existing files) and ALLOC (for new or revised files).

(2) ALLOC uses a set of statement functions to allocate available memory to the CLAS file and the HIST file. An iterative computation of available memory expands three file parameters until the limit is met. User requests to allocate space for extra prototypes (variable NE) are honored first; requests for histogram intervals (variable NBUC) are honored next; then, requests for space for prototype husk entries (variable MAXKV) are filled. If changes are made, user approval is requested. Disapproval aborts the module.

(3) NEXCLA controls optional processing of each class of data. Table B-III defines its input controls. Embedded in this routine is the mechanism which allows direct user assignment of feature vectors to the husk of a class. Multiple passes through each FEAT file record are possible through an option in this routine. General control inputs include plotting parameters, as well as processing function selections. The primary output of the routine (variable NEXC) identifies the class data set about to be processed.

(4) PHUSK is a support routine which uses KERGET to access and print the husk list associated with a prototype.

(5) KERGET is a support function which uses RIX to extract consecutive husk vectors from the CLAS file entry for a given prototype. The next stored husk entry is returned on successive calls in which the data class specification remains the same.

(6) KERPUT is a support routine through which a short list of husk vector numbers can be inserted into the linked list of husk vector numbers which is maintained in the CLAS file.

(7) CLASSX is the primary control routine within DEFINE. If the user has selected an initialization process, CLASSX initializes the CLAS file using INITC and ADD. If a follow-on process has been requested, CLASSX establishes prototype locations within the CLAS file, and allows revision of those addresses. The major cycle of CLASSX provides FEAT records to CDEFI, FANDER, SHUCK or PICKER as requested by control parameters. SHUCK and PICKER are dummy exits for either automatic or interactive graphic assignment of feature vectors to the husk of a class. In addition to this control process, CLASSX updates the current HIST record whenever CDEFI has processed. Both an exit trace, and a trace of internal computations are embedded in this code.

(8) FANDER produces a plot of feature vector components. The ordinate of this plot can be scaled according to three options requested by NEXCLA. See Table B-III. The abcissa of this plot consists of a set of discrete locations, one for each dimension

of the feature space. Plotting produces a set of points for each feature vector. These can be connected to suggest the character of the individual feature vector. All vectors within each data block of a given FEAT file record can be accessed and plotted by FANDER. The effect is that of a heavily overlayed set of line graphs which suggest in a single display the degree of correlation and the variance in all feature dimensions. The combination of this picture and either a listing of vector components or the PICKER routine supports shucking unreasonable feature vectors from the FEAT file set used for prototype generation. Figs 32 and 33 provide samples of FANDER output.

(9) CDEFI generates prototypes. At each call, CDEFI processes one block of the current FEAT file record. At each exit a prototype exists within the CLAS file which reflects all feature vectors processed to that exit. KERGET is used to reject from this process any feature vectors assigned to the husk of the class. A HIST file record is updated at each call to CDEFI and is available for use at each exit. Both an exit trace and a log of intermediate calculations are supplied.

(10) SETLIM accesses the FEAT file trailer record to obtain maximum and minimum component values established for each dimension of the feature space by CREATE. It then uses ADD to establish a CLAS file entry for this data, and updates the CLAS file. These global component limits are used within TRYOUT and FORMAT in order to scale feature components into the byte sized

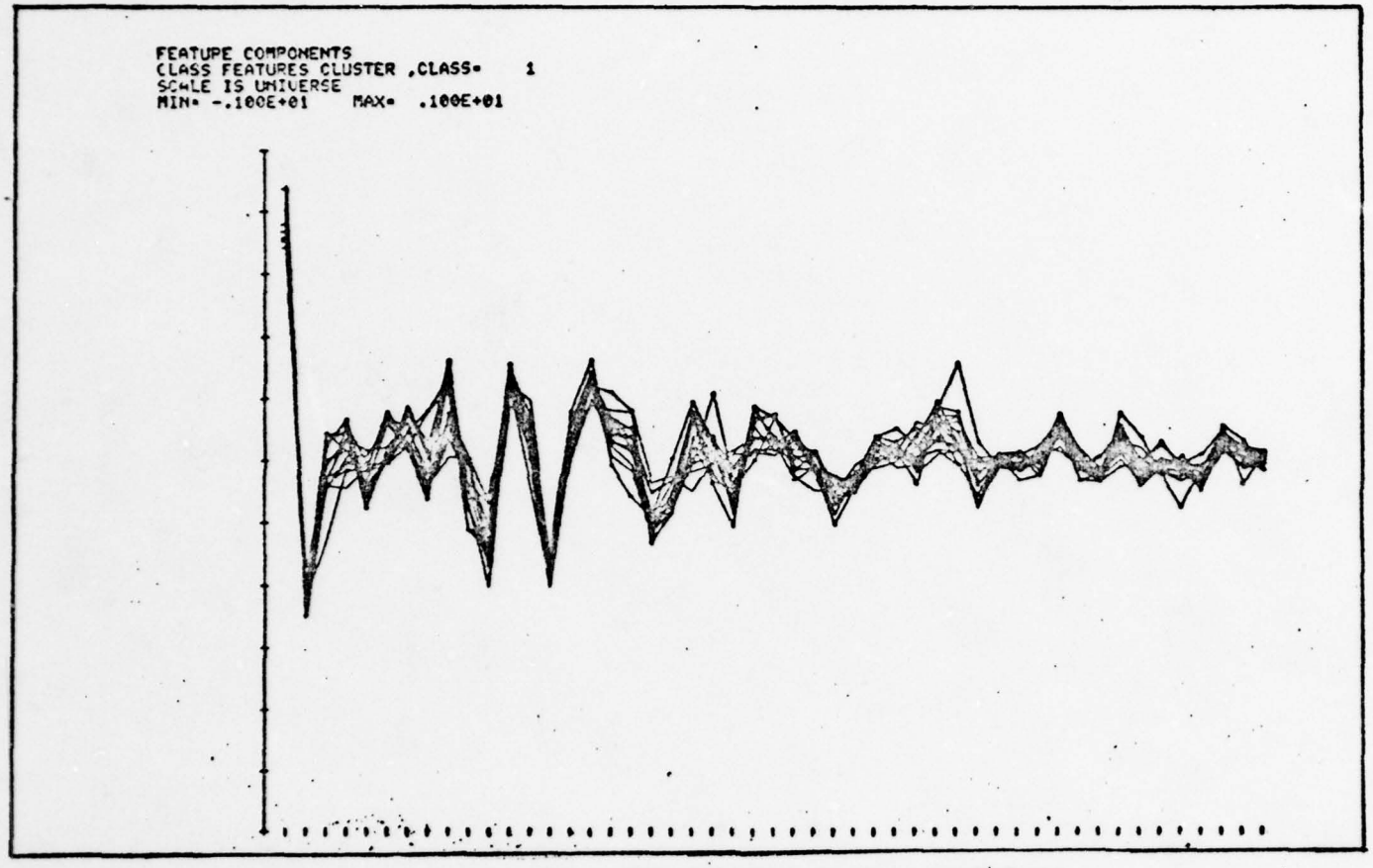


FIG 32. FEATURES CLUSTER PLOT - SAMPLE(A)

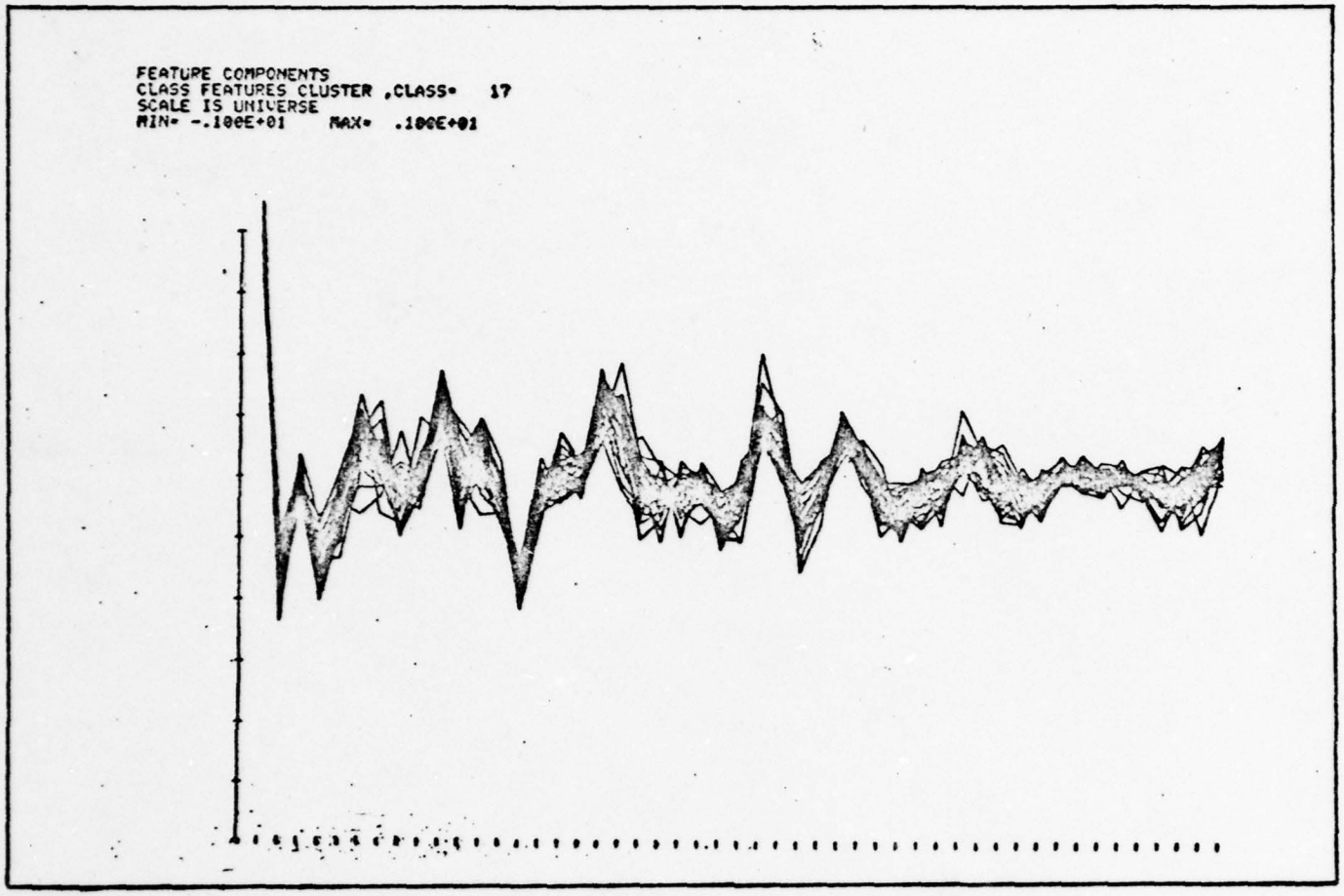


FIG 33. FEATURES CLUSTER PLOT - SAMPLE(B)

range (0-255) required for microprocessing.

DEFINE processing has three major paths. The initialization path is followed when selected as an option at module start. Prototypes can be initialized only as a complete set one to one with the FEAT file. When a CLAS file is initialized the last record of the FEAT file is entered into the CLAS file. This record contains scale factors for the feature space which are used in other DEFINE paths and in both TRYOUT and FORMAT. Thus an initial CLAS file is a pre-requisite for all other DEFINE processing paths. The regeneration path supports selection of husk vectors and allows definition of a new prototype without processing those vectors. Prototype husks are stored in the CLAS file; this regeneration process can be a heuristic iteration. When this path is followed, specific prototypes may be selected for revision. A given class of feature vectors (i.e., a record from the FEAT file) may be completely processed in repetitive iterations. The third path allows generation of asymmetric prototypes; its processing parallels that of the regeneration path. A CLAS file is output whenever DEFINE is run. A variety of selectable outputs may be written to the LOGF file. Appendix K contains a sample LOGF file produced by DEFINE. Table C-III briefly describes the contributions of each routine to this output. A list of terminal outputs is presented in Table C-II. Output messages are described in each table in their approximate order of appearance during program execution.

TRYOUT. This module performs a trial classification of the feature vectors within a given FEAT file. It can be used to evaluate the acceptability of a given CLAS file. Additionally, it can be used to estimate the relative merit of individual feature dimensions and to construct from the original feature space a subspace within which classification is optimal. The primary output of this module is a summary statement of classification error rate. Input options extend this statement to a confusion matrix format, and to a set of error rates for each of a set of vested subspaces of the original space. A secondary output is a revised version of the input CLAS file. This revision reflects both scaling and zapping of prototype components. These and the other functions of TRYOUT are described below with the subroutine which implements them. Fig 34 presents a data flow chart for TRYOUT. Fig 35 presents a structure diagram.

TRYOUT is composed of seven major functional subroutines. These are DEFT, MERIT, SUBSET, FIGM, EVAL, LOOK and DOCU. The sequence of subroutines called as TRYOUT is executed is listed in Table VII. The input and output parameters for unique TRYOUT subroutines are defined in Table D-III. Each is synopsized below.

(1) DEFT initializes TRYOUT. User inputs are obtained to define selectable options. Table B-VI defines controls input by DEFT. Both CLAS and FEAT files are opened, and memory allocation parameters are set and checked against the system

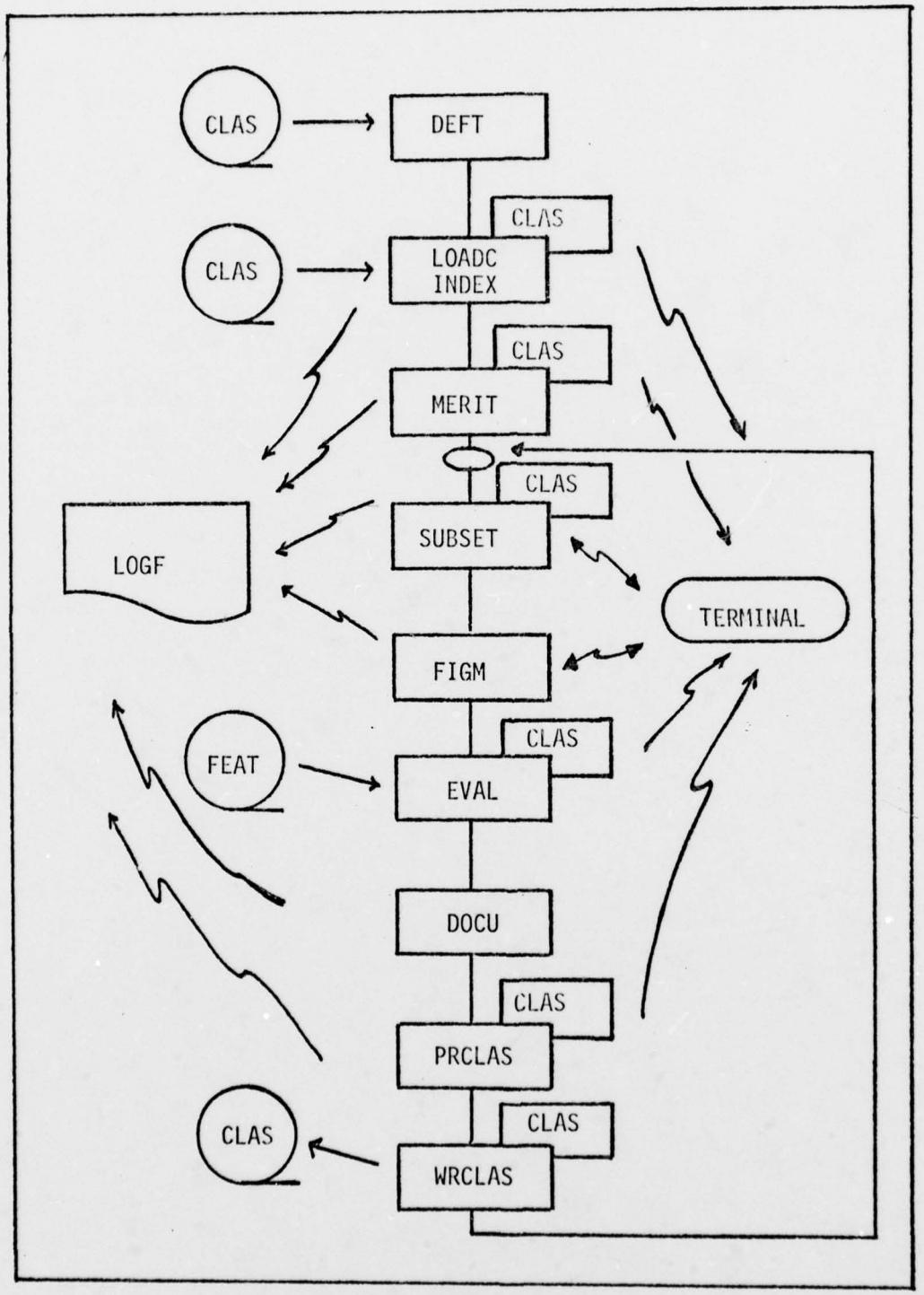


Fig. 34. TRYOUT Data Flow

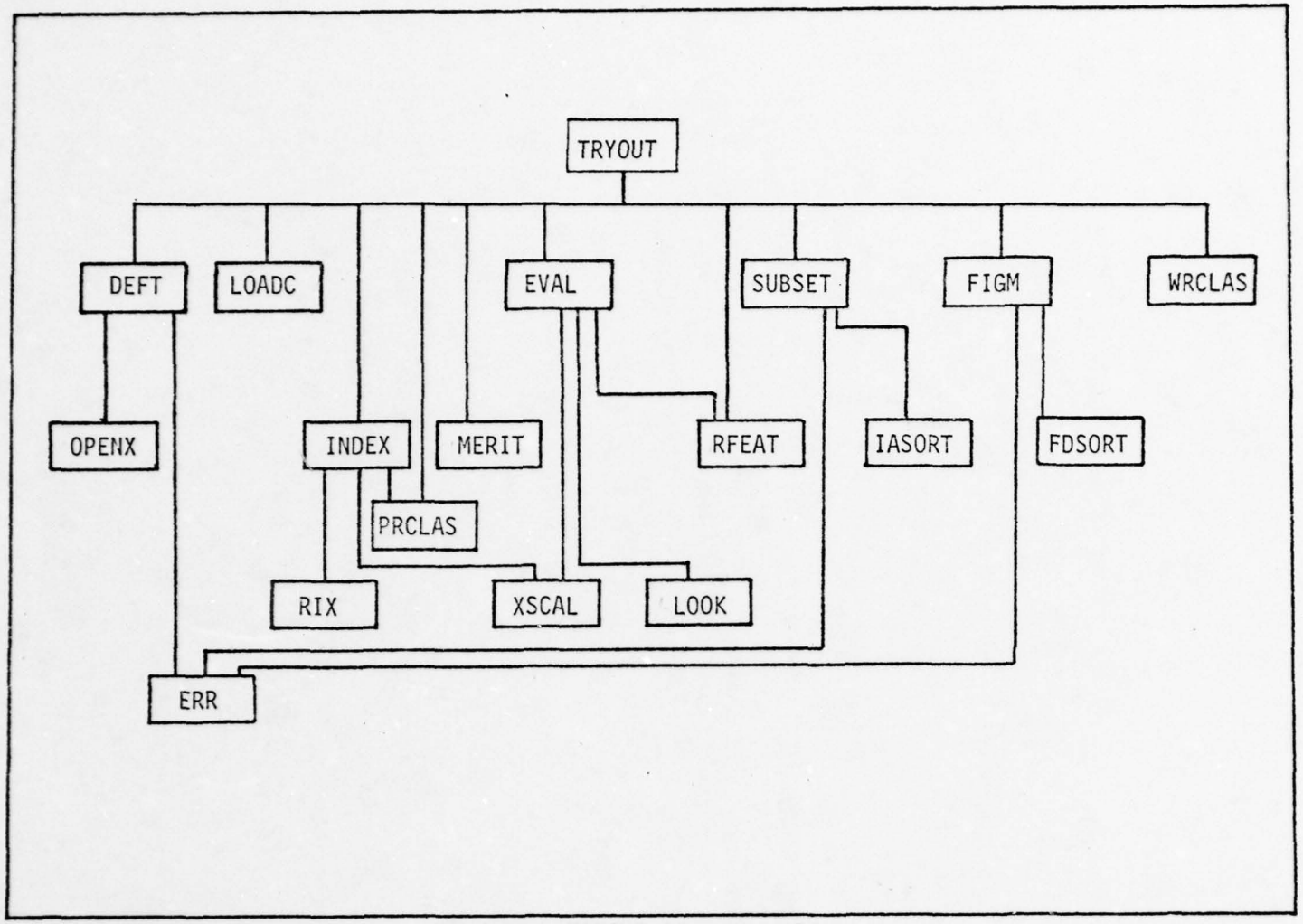


Fig. 35. TRYOUT Structure Diagram

TABLE VII

Sequence of Calls in TRYOUT Process

Routine	Description
TRYOUT	Classifies FEAT records against CLAS file
DEFT	Defines module control parameters
OPENX	Opens FEAT and CLAS files
ERR	Echoes error prompt to terminal
LOADC	Loads CLAS file
INDEX	Builds special index to CLAS file
RIX	Reads CLAS file index
XSCAL	Scales prototype into specified value range
PRCLAS	Prints CLAS file
MERIT	Computes a figure of merit for each dimension
SUBSET	Controls prototype component zapping
LOADC	Loads CLAS file
INDEX	Rebuilds special index to CLAS file
RIX	Reads CLAS file index
XSCAL	Scales prototypes as specified
PRCLAS	Prints CLAS file
ERR	Echoes error prompt to terminal
IASORT	Sorts zap tags to ascending order
FIGM	Requests subspace id or subspace tags
FDSORT	Sorts subspace tags into descending order
EVAL	Classifies a given feature vector against CLAS
RFEAT	Reads FEAT file
XSCAL	Scales FEAT vectors into given range
LOOK	Computes feature subspace error rates
DOCU	Documents classification error rates
PRCLAS	Prints CLAS file
RFEAT	Reads FEAT file
WRCLAS	Writes CLAS file to disk

*Underlined routines are unique to TRYOUT

limit. DEFT sets 'NAMES', the common block of file names used by TRYOUT.

(2) MERIT computes five sets of figures of merit for the feature components represented in the CLAS file prototypes. Three of these are intermediate computations used nowhere else. Feature evaluation algorithms (see chapter 4) are used to compute the output sets of merit figures.

(3) SUBSET is a control subroutine which requests user inputs to direct the process of feature zapping. This process expands specified feature deviation values within a given prototype. The effect is elimination of the specified feature component from the classification process. Table B-IV defines user inputs to SUBSET. Figs 18 to 23 present a sample execution of TRYOUT showing some of these inputs.

(4) FIGM is a control subroutine through which the user selects which set of merit figures are to be used in production of an extended set of classification error rates. Table B-V specifies control inputs to FIGM. A list of feature dimensions, considered to define a set of nested subspaces, is passed from FIGM to LOOK which computes the subspace error rate statements. Refer to Appendix K for a sample operation of FIGM.

(5) EVAL is the core subroutine of TRYOUT. It controls reading of the FEAT file and classifies each feature vector within this file against prototypes within the CLAS file. The BOX80 decision rule is implemented so as to admit prototypes

whose components have been scaled into the range 0-256. This allows simulation of the processing within DECIDE, the 80/20 classifier. Additionally an option allows the user to elect use of the Euclidean norm rather than the Sup norm within the decision process. EVAL outputs a summary of error rates and a confusion matrix. It also controls execution of LOOK.

(6) DOCU is an output format routine. The summary performance error rate, the confusion matrix, and the list of subspace error rates are printed by DOCU.

(7) LOOK analyzes each distance vector computed within the classification algorithm. The components of this vector are re-ordered according to the list of subspace tags provided by FIGM. Then each nested subvector is classified within its subspace and error rates are recorded for later output by DOCU. There is a tight interface, that is, there are no subroutine parameters and there is significant interlacing of common blocks, tieing this routine to EVAL and to DOCU. This, since LOOK is called inside the inner most loop of TRYOUT.

TRYOUT processing consists of an initialization sequence and an evaluation cycle. In the former, DEFT, OPENX, INDEX and MERIT establish processing options and parameters, load and scale the CLAS file if opted, and compute MERIT figures. Control inputs to this sequence are shown in Table B-VI. In the latter, FIGM, SUBSET, EVAL, LOOK, and DOCU allow the user to modify the feature

dimensions used in the classification process, and then perform and document that process. A revised CLAS file is output at end of job whenever the CLAS file has been zapped via SUBSET. A variety of selectable outputs may be written to the LOGF file. These are triggered by the standard control option (L,T,Y) and by the TRYOUT control option (C,A). Appendix K contains a sample LOGF file produced by TRYOUT. The contributions of each routine to this LOGF output are summarized in Table C-IV in the approximate order of their generation.

FORMAT. This module produces several formats of data within each of the BOX80 system files. Its primary purpose is the production of the PROT and FVEC files in hexadecimal paper tape line format for input to 8080 microprocessor systems. Secondarily displays of CLAS, FEAT and HIST records are produced on the TEKTRONIX 4014 terminal screen. The module is designed to produce two display formats. The strip chart format, which is only stubbed into the code, is intended to allow precise examination of ordinate and abcissa data values for individual prototypes, feature vectors and feature histograms. The picture format presents a top level three-dimensional presentation of global data variation within sets of prototypes, feature vectors and feature histograms. These and the other functions of FORMAT are described below with the subroutine which implements them. Fig 36 presents a data flow chart for FORMAT. Fig 37 contains a structure diagram, while Tables B-VII and B-VIII show input

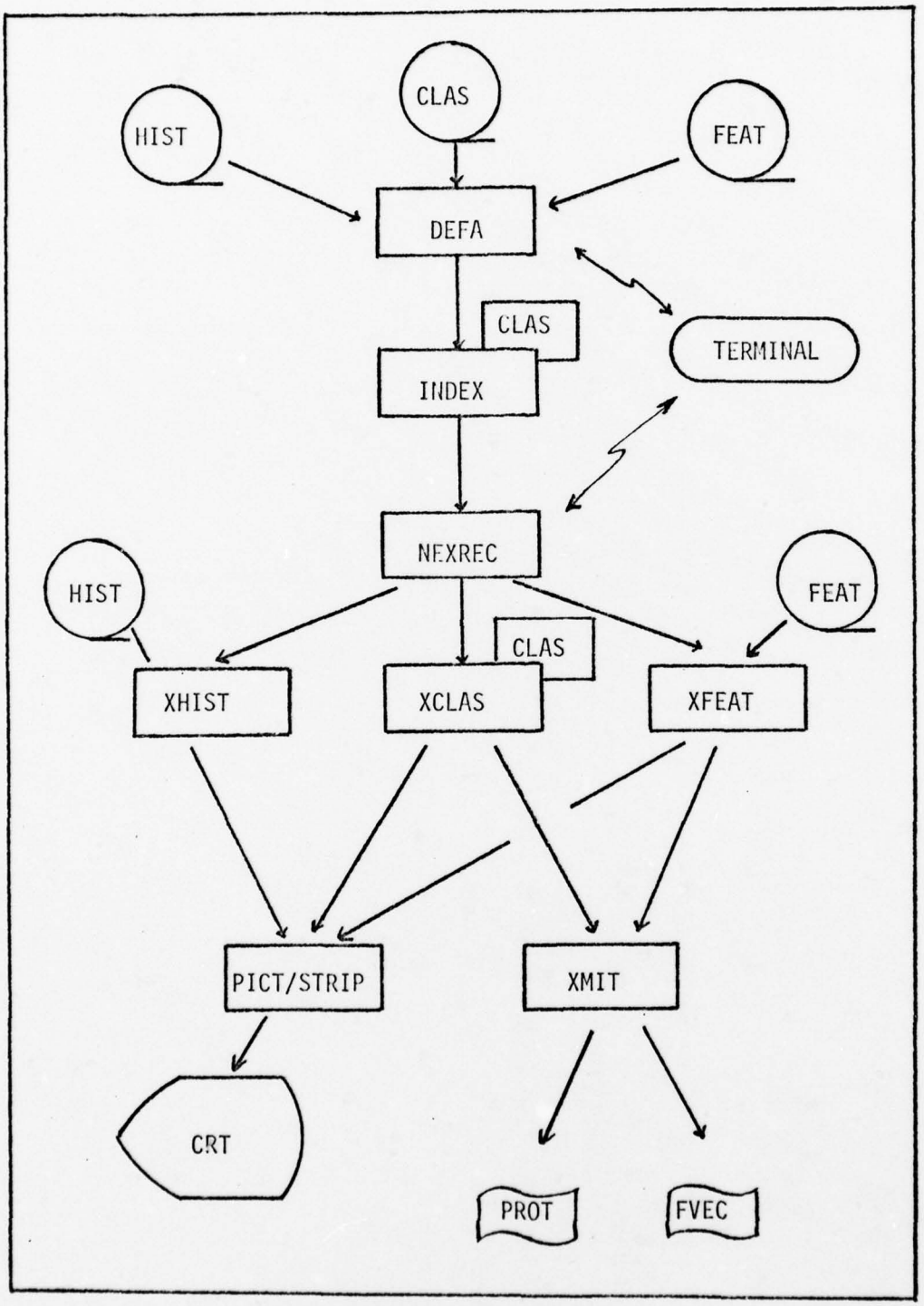


Fig. 36. Format Data Flow

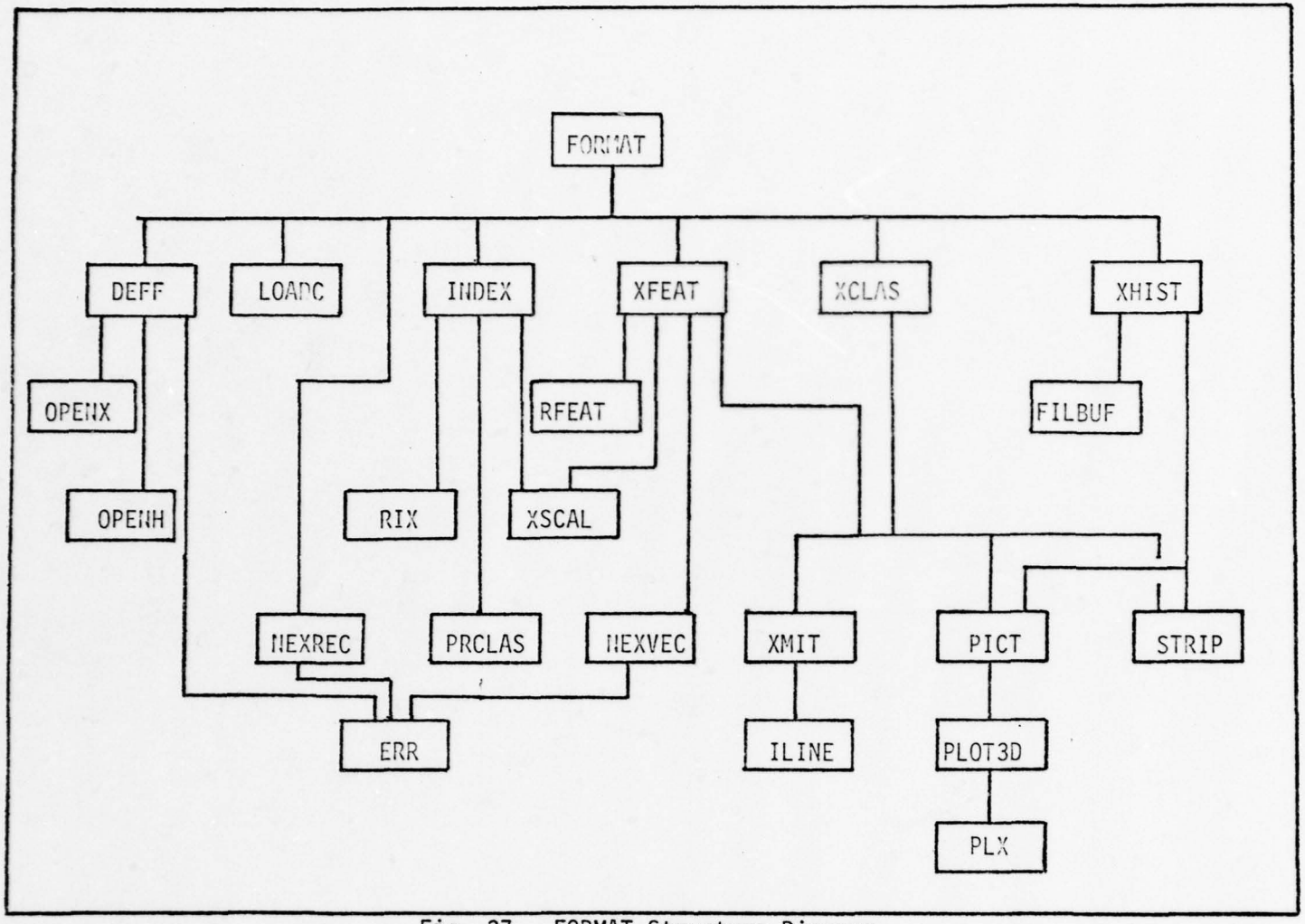


Fig. 37. FORMAT Structure Diagram

TABLE VIII (1/2)

Sequence of Calls in FORMAT Process

Routine	Description
FORMAT	Produces output format from BOX80 file
DEFF	Defines module control parameters
OPENX	Opens CLAS file, and FEAT file if opted
OPENH	Opens HIST file
ERR	Echoes error prompt to terminal
LOADC	Loads CLAS file
INDEX	Builds table to index CLAS file; may scale CLAS
RIX	Finds CLAS file index entries
XSCAL	Scales vector components to stated range
PRCLAS	Prints CLAS file
NEXREC	Requests user input of next data class
ERR	Echoes error prompt to terminal
XFEAT	Controls processing each FEAT file record
RFEAT	Reads FEAT file record blocks
NEXVEC	Requests user choose specific FEAT vectors
ERR	Echoes error prompt to terminal
PICT	Sets up for 3-D plot
PLOT3D	Hidden line routine draws feature vectors
PLX	Emulates CALCOMP PLOT routine
STRIP	Stub for feature vector strip chart function
XSCAL	Scales vector components into stated range
XMIT	Drives hexadecimal line format
ILINE	Produces hexadecimal line output
XCLAS	Controls processing each CLAS prototype
XMIT	Drives hexadecimal line format
ILINE	Produces hexadecimal line output
PICT	Sets up for 3-D plot of prototype data
PLOT3D	Hidden line routine draws prototype boundaries
PLX	Emulates CALCOMP PLOT routine
STRIP	Stub for feature vector strip chart function

TABLE VIII (2/2)

Sequence of Calls in FORMAT Process

Routine	Description
XHIST	Controls processing HIST file records
RHIST	Reads HIST file records
FILBUF	Builds buffer for 3-D plot
PICT	Sets up for 3-D plot of histograms
PLOT3D	Hidden line routine draws nistograms
PLX	Emulates CALCOMP PLOT routine
STRIP	Stub for histogram strip charting

*Underlined Routines are unique to FORMAT

options and controls. Table C-V summarizes outputs in the approximate order of their appearance on the LOGF file. Table VIII shows the sequence of subroutine calls executed in operation of FORMAT.

FORMAT is composed of five major functional routines and several unique supporting routines. These are described in the paragraphs that follow. The input/output parameters for these unique routines are defined in Table D-IV.

(1) DEFF initializes control parameters for FORMAT and allocates available memory to buffers and tables. Input options allow selection of a file data source and choice of a processing option. FEAT, CLAS, and HIST files may be input. Process options are transmit, picture and stripchart. The selected source data file(s) are initialized via subroutine call from this module.

(2) XCLAS directs processing of CLAS file data. This routine has three data paths. When the transmit option has been selected, XCLAS formats a PROT file with the non-zapped components of selected prototypes and prepares a count of the dimensionality of the prototype space represented by the PROT file. If either stripchart or picture options have been chosen, a buffer is filled and output to the appropriate routine when full.

(3) XFEAT controls the processing of FEAT file data. A special routine (NEXVEC) allows selection of specific feature vectors. These are either output for transmission as hexadecimal data lines, or loaded into the buffer used for data

display. When transmission has been opted, only values of non-zapped features are processed.

(4) XHIST handles the flow of HIST file data to the display buffer. A special subroutine (FILBUF) does the actual movement of data items. A utility routine (RHIST) provides access to the HIST file. When a DIST file (a single record HIST file produced by CREATE recording universe distributions) has been input, XHIST sets special processing parameters.

(5) XMIT is the controlling driver for the hexadecimal format routine.

(6) PICT is the controlling driver for the 3-D plot routine. It requests and sets plot scaling parameters, controls repetitive displays, and initializes TEKTRONIX graphics.

(7) STRIP is the stub for a routine which should initialize TEKTRONIX graphics, and label and output a set of stripchart plots with scaled axes.

(8) FILBUF passes HIST record data to STRIP and PICT. It allows a LOGF file printout of feature distributions and statistics as well.

(9) NEXREC is a control subroutine through which the user selects classes of data for processing.

(10) NEXVEC is a control subroutine through which the user identifies (sets of) feature vectors for processing.

Format processing begins with the system standard initialization during which the selected data file is opened. A major cycle through each data class can be automatic at the request for each desired class. Classes of data must be processed in ascending order by class number. When the transmission option is elected, only one address may be specified for the output PROT or FVEC file. However, when picture or stripchart options are selected multiple display outputs are possible so that differently scaled presentations can be viewed. Similarly, when FEAT files are processed, a given data class may be processed repetitively so that different sets of feature vectors may be output. This should aid in the selection of a kernel of feature vectors from which to define a class archetype. Output to the LOGF file is minimal, consisting mainly of journal entries of user inputs. However, a format print-out of feature statistics is provided. Appendix K contains a sample LOGF file produced by FORMAT.

Classifier Segment

This segment consists of two modules. These are TAPEIN and DECIDE. The former is a support module. The latter implements the BOX80 system classifier. They are described in the following subsections.

TAPEIN. This module loads PROT and FVEC files into microprocessor RAM in order to set up data buffers for execution of the DECIDE module. The module is dependent upon service

routines within the SBC 80/20 ISIS 1.0 monitor. It's design is based upon the ISIS routine which implements the SBC 80/20 "R" command (Ref 19). Output from the module is simply a block of RAM locations which are loaded with the data contained on an input cassette tape. Timing and control variations between paper tape and cassette tape readers necessitated the module.

TAPEIN is used as a utility of the SBC 80/20. It is executed according to procedures detailed in Table B-IX. Data are input to the TAPEIN module on a cassette tape produced by copying a PROT file generated by the Interpreter Segment. The procedures for generating this file are shown in Table B-X.

DECIDE. This module classifies feature vectors. It executes within SBC 80/20 RAM and references RAM locations to obtain both class definitions and pattern feature vectors. Outputs from this module are a decision by decision record of class assignment, and a summary count of correct and incorrect decisions. The module is designed to be a model, and not to be a packaged subroutine. Thus, its initialization requires the user to manually set 80/20 RAM with control values. For interpreter testing these initializations are duplicated by references to assembler symbols, which should be set before assembly. These initial values specify the dimensionality of the feature space (JD), the number of pattern classes (IC), and the number of feature vectors in the data block to be processed (LB).

DECIDE is intended to be used as a supporting process within one of a pair of microprocessors which communicate via a common buss and a central RAM. The primary processor acquires data, and generates feature vectors. As each vector is produced, the secondary processor is interrupted, and the vector is placed in RAM. The secondary processor is triggered when the first vector is entered into the RAM data block. It continues to execute DECIDE, producing classification decisions, until this data block is empty.

A priori knowledge of test feature vector classification is reflected in DECIDE output. The score keeping element of the DECIDE process should be deleted in any actual implementation. This code is located in code paragraph 0A5, and is shown in the flow chart in Fig 38. Figs 39 and 40 present the data flow and structure within DECIDE.

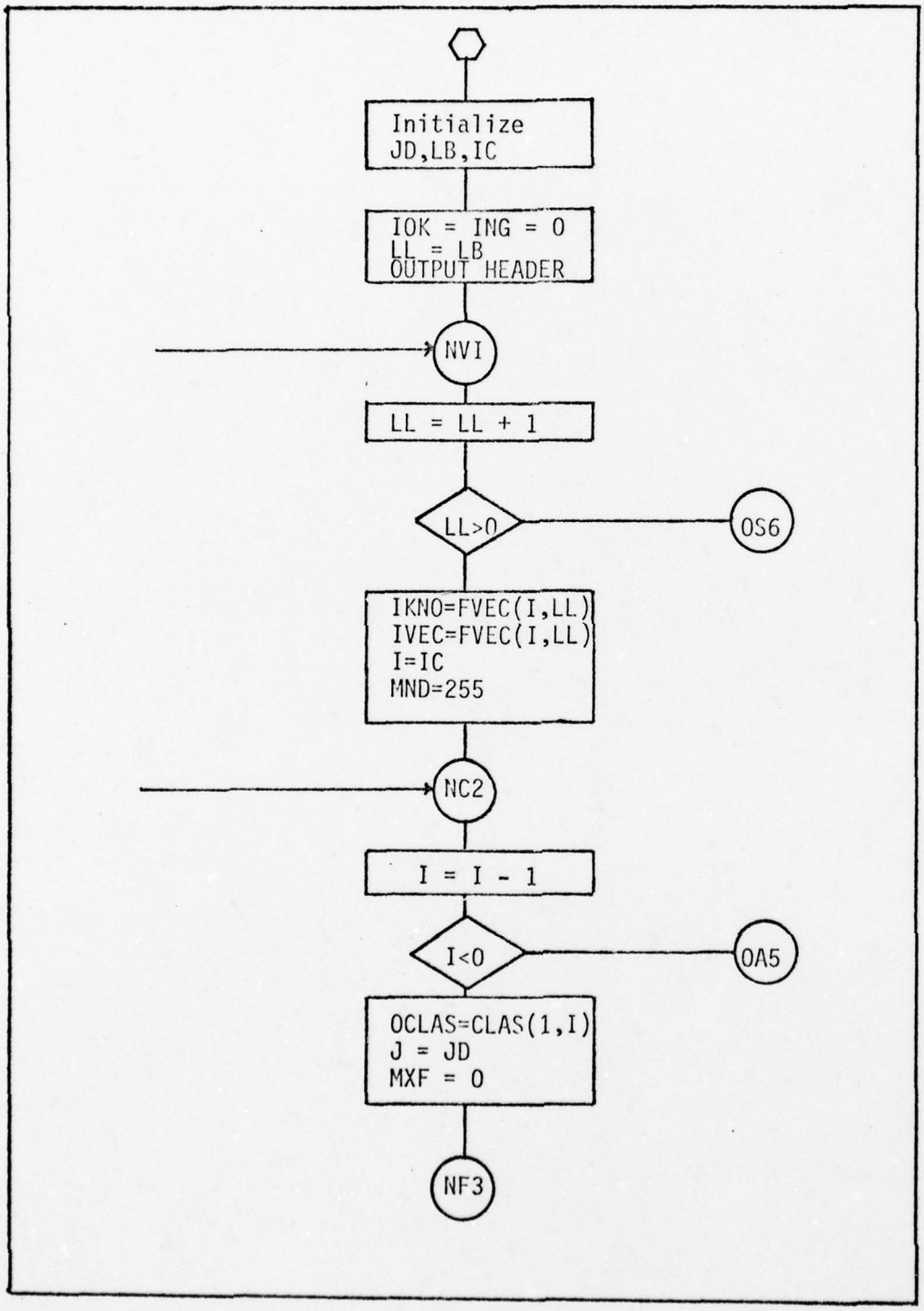


Fig. 38. Flow Chart for DECIDE (1/3)

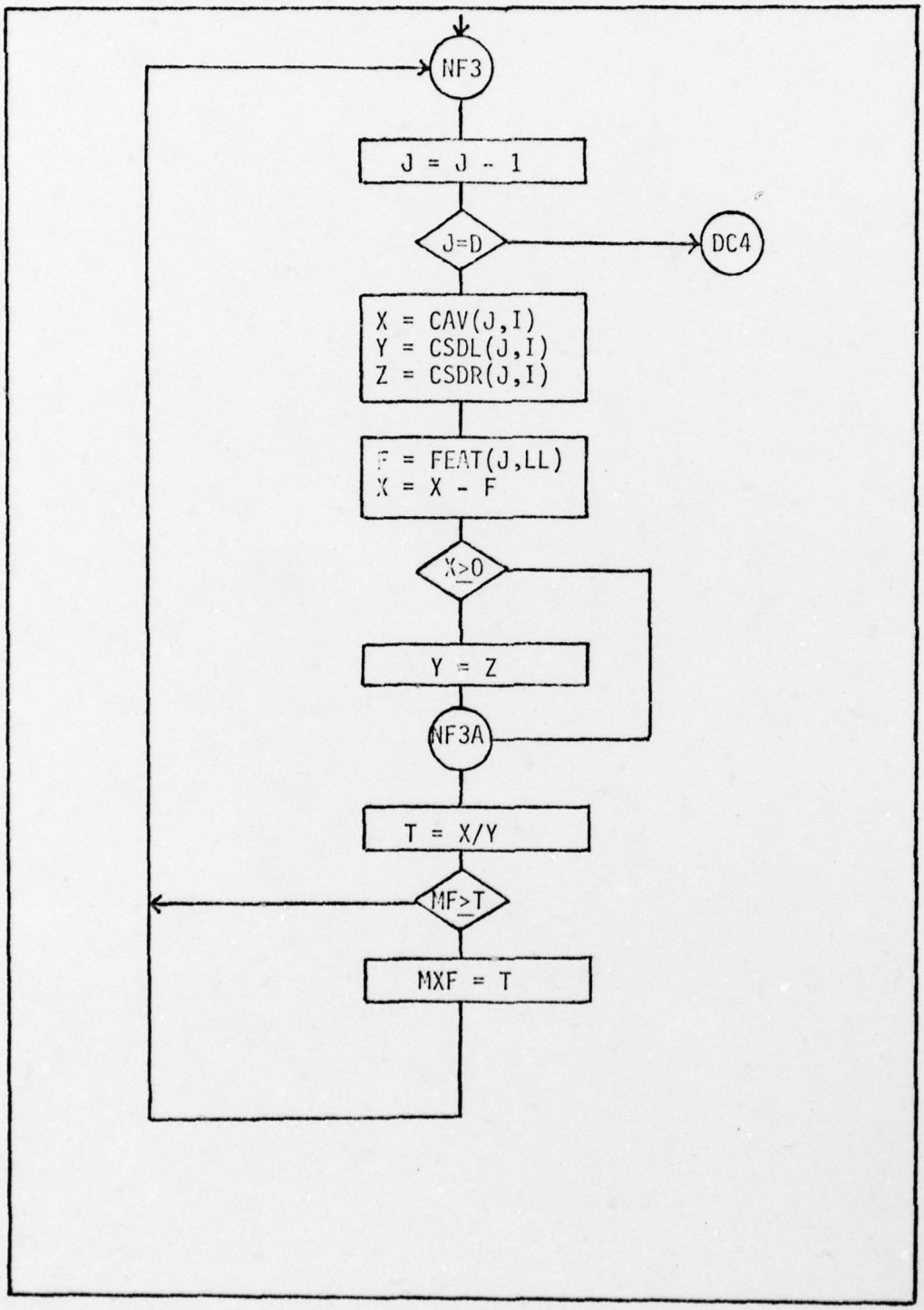


Fig. 38. Flowchart for DECIDE (2/3)

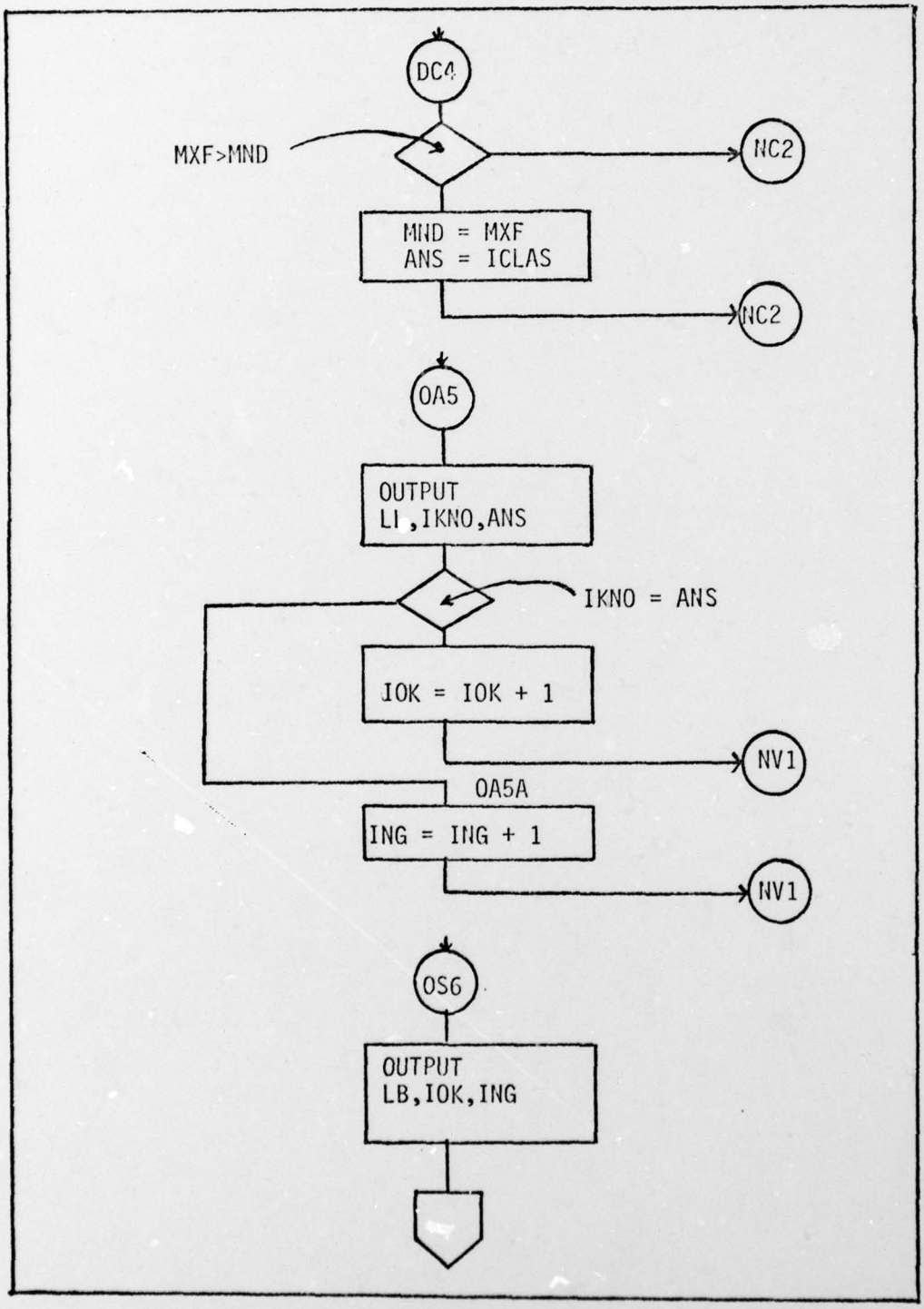


Fig. 38. Flow Chart for DECIDE (3/3)

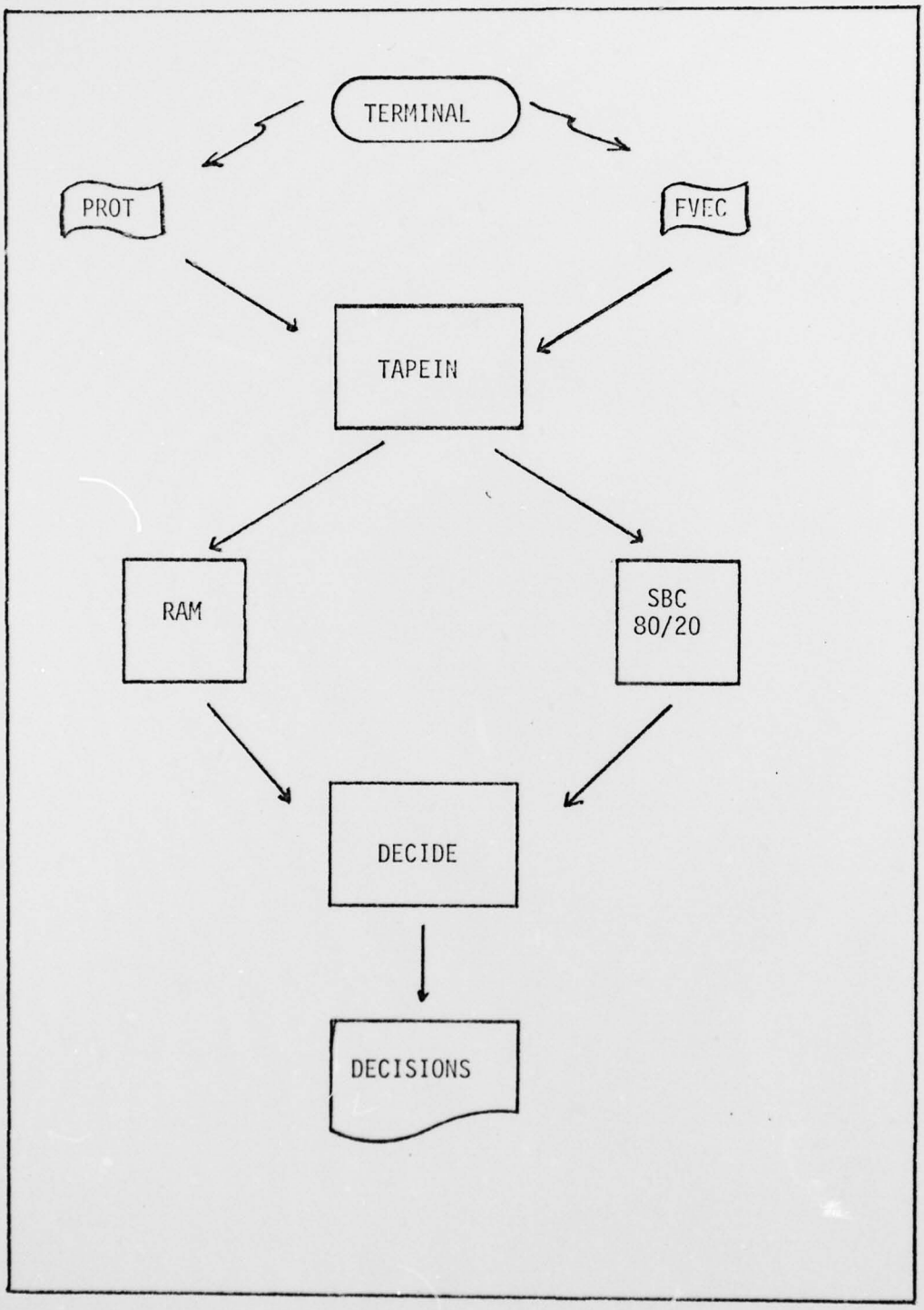


Fig. 39. Classifier Segment Data Flow

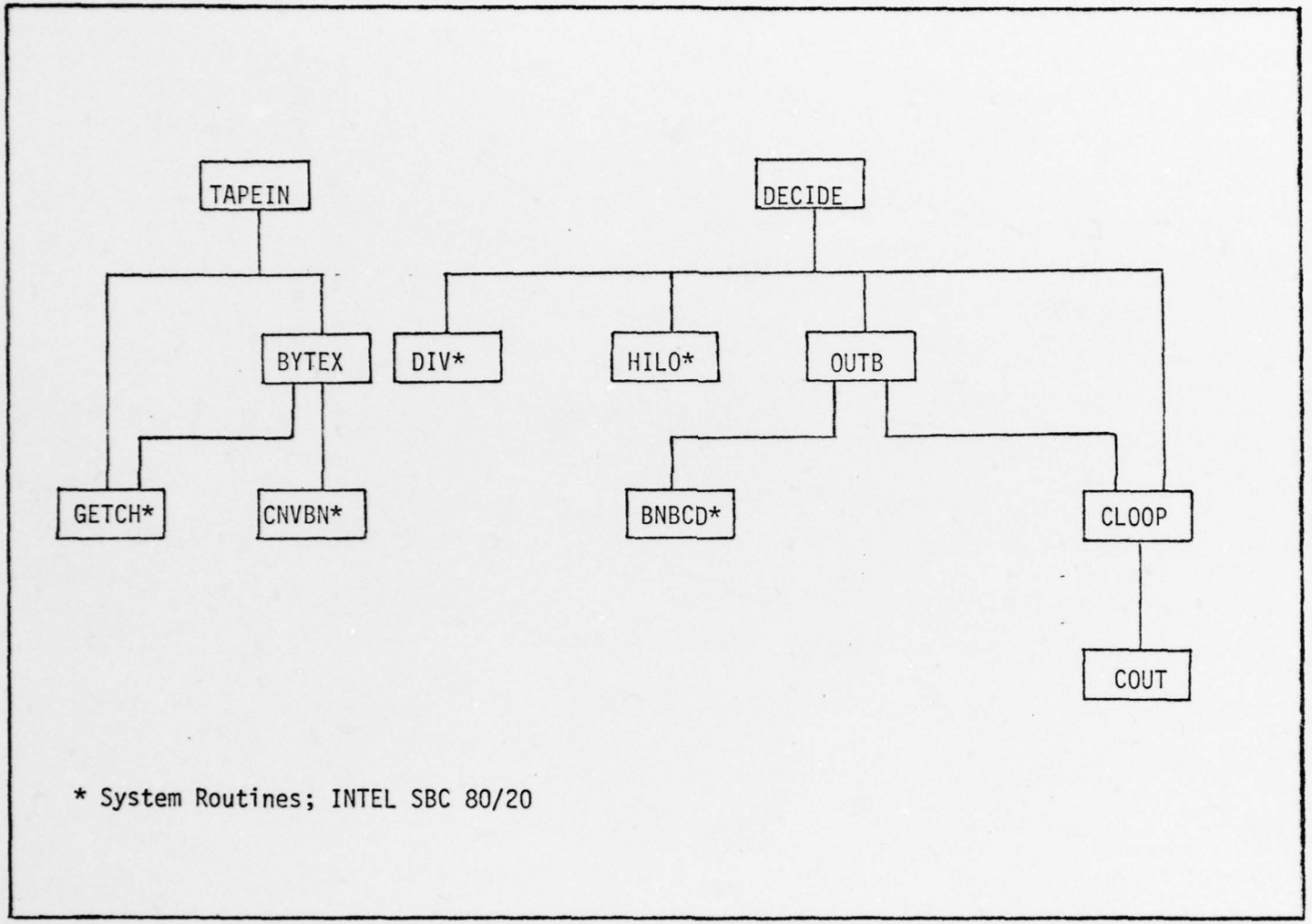


Fig. 40. DECIDE Structure Diagram

VI. Conclusions and Recommendations

This thesis has presented a development system for microprocessor based pattern recognizers. Two system segments were implemented. These satisfy the functional requirements established for the system. The algorithms developed for the system were defined and were illustrated in the preceding chapters. The design of the computer program modules which comprise the system was described in Chapter V. A performance evaluation was provided for the system through a series of benchmark experiments. Specific conclusions and a set of recommendations are now provided in the following sections.

Conclusions

The BOX80 system provides a framework for experimentation. It can be used to configure a pattern classifier which forms one node of a two-part microprocessor based pattern recognizer. The Classifier Segment of the BOX80 system has been tested by simulation. This testing has shown that the classifier algorithm can indeed produce recognition decisions with an acceptably low error rate. The Interpreter Segment of the BOX80 system has been demonstrated by experiment. Class defining structures have been generated and trial performance has been measured. The contrast

of this performance to independent experiments using the same data has shown that the Interpreter Segment can support accurate pattern recognition. The algorithms used in this latter segment include a non-parametric, weighted, minimum-distance classification procedure, and a manually controlled feature selection technique.

The classifier algorithm was shown to be capable of performance approximately equivalent to that obtained from OLPARS' and SPSS' algorithms. This performance in fact exceeds that of previous AFIT experiments with benchmark data sets (Refs 24, 33) and verifies simulated alphabet classification error rates projected by Tallman (Ref 35). Although suboptimal, this classifier algorithm is very efficient. Existing AFIT programs, and even the SPSS system, require far more memory for class defining data structures than the BOX80 classifier requires. One execution time comparison showed a 2:1 run time improvement. The concept of microprocessor development relies upon the use of byte-scaled features. Experiments with both the FOBW and the alphabet data showed a less than one percent average increase in errors when the classifier algorithm operated on these byte scaled integer values.

The feature selection algorithm was shown to be comparable to the OLPARS procedure. Although possibly more difficult to use, the BOX80 procedure is more flexible than that of OLPARS.

The minimum error rate produced by the BOX80 system is equivalent to that produced with OLPARS' NMV classifier. This comparison is, of course, highly data dependent. The BOX80 system feature selection algorithm chose a best series of nested feature subsets for classification of the alphabet data. The series of associated error rates decreased monotonically and asymptotically. The error rate for each of these nested feature subsets was lower than the error rate for every other tested feature subset of the same size. The final subspaces selected for the alphabet and the FOBW data sets each produced error rates less than or equal to the lowest error rates obtained by previous AFIT experimenters. Note that these previous experimenters used two and seven times as many features for their lowest error rates as were used in the comparable BOX80 tests.

The Interpreter Segment of the BOX80 system embodies processing capabilities which have not yet been fully explored. The CREATE module has options for input data transforms which were not experimentally evaluated. The DEFINE module has the necessary data structure to support editing the training data set so as to define class structures based on analytically selected class kernels. Subroutine stubs are indicated but not provided for an automatic editing capability. The TRYOUT module allows selection of partially disjoint feature subsets for each data class. Data processing structure for generation of rejection rates exists. The FORMAT module has indicated but not provided subroutine stubs for strip chart graphics presentations of histogram,

and feature vector data. None of these capabilities was required of the Interpreter Segment. The conclusion here is that a significant capacity for enhanced capability is deliberately designed into the system.

Finally, the BOX80 system is transportable as required. This fact is not explicitly shown. However, ANSI code conventions were followed. Design is modular and data structures are sized by the user. The use of independent modules related by standard files supports the transportability of this code. This transportability and the economy of its algorithms make the BOX80 system a potentially valuable tool for the development of microprocessor based pattern recognizers.

Recommendations

A host of general suggestions are possible. One outweighs all others. The system should be used in an experimental development of a waveform pattern recognizer. The systems design for this experiment should address the all-important problem of generating a design data sample which adequately represents the pattern environment. Local research facilities have supported experiments of this type which have processed electrocardiographic data. Because of this ready availability, this data should be used for a first experiment with the BOX80 system. A list of more specific recommendations follows.

(1) Error rates achievable with the asymmetric classification option should be experimentally compared to those achievable with the symmetric process.

(2) The TRYOUT module should be modified to experiment with the use of reject boundaries. A constant boundary level should be used for all classes at first. Then unique boundaries should be used for individual classes.

(3) The capabilities of the DEFINE module for edit selection of husk feature vectors should be explored.

(4) A new module, MODIFY, should be produced to investigate formation of synthetic classes. These should be formed between classes whose members are easily mistaken as indicated by confusion matrix output. This module should present interclass distance measures in graphics and tabular form. These measures should be designed to qualify the effect of selecting kernel patterns on the variances and dispersions of individual features.

(5) The DEFINE module should be modified to investigate mode based class defining structures.

All of the above experimental modifications should use the alphabet data set produced by Sponaugle as a standard test data set. The value of the Fourier transform features recorded on that data set should be further qualified by a classification experiment using the 81 space vectors generated by Sponaugle.

Bibliography

1. Bouvier, Ronald D. Seismic Pattern Recognition. MS Thesis, Wright Patterson AFB, Ohio: Air Force Institute of Technology, December 1972 (AD 757 877).

2. Box, G.E.P., and J. Ledolter. Topics in Time Series Analysis, Technical Report No. 446, Madison Wis.: University of Wisconsin, December 1975 (ADA026311).

3. Chen, C. H. "On a Class of Computationally Efficient Feature Selection Criteria," Pattern Recognition, 7:87-94 (June 1975).

4. Chen, C. H. A New Look at the Statistical Pattern Recognition, Technical Report No. EE-77-4, North Dartmouth, Mass.: Southeastern Massachusetts University, August 1977.

5. Connell, D. B., et al. MULTICS OLPARS Operating System, RADC TR-75-271 Griffiss Air Force Base, N.Y.: Rome Air Development Center (ISCP) September 1976 (ADA034393).

6. Conte, S.D. and C.E. Boor. Elementary Numerical Analyses, New York: McGraw Hill Book Col, 1972.

7. Control Data Corporation, FORTRAN Extended Reference Manual, 60305600, Minneapolis, Minn.

8. Cover, T. M. "The Best Two Independent Measurements Are Not the Two Best," IEEE Transactions on Systems, Man, and Cybernetics, SMC-4: 116-117 (January 1974).

9. Das Gupta, S. Some Problems in Statistical Pattern Recognition, Technical Report No. 258, Minneapolis, Minn.: University of Minnesota, January 1976 (ADA035048).

10. Feucht, D. "Pattern Recognition: Basic Concepts and Implementations," Computer Design, 57-68, December 1977.

11. Fix, Evelyn and J. L. Hodges. Discriminatory Analysis: Nonparametric Discrimination, Project 21-49-004, Report 4, Randolph AFB, Texas: USAF School of Aviation Medicine, February 1951.

12. Godwin, H. J. Inequalities on Distribution Functions, London: C. Griffith and Co., 1964.

13. Gonzalez, R. C. and J. M. Harris. Feature Extraction and Recognition of Two-Dimensional Data by the Method of Moments. Technical Report ONR-CR215-228-2, Knoxville Tenn.: University of Tennessee, January 1977 (ADA037445).

14. Hall, Charles F. The Analysis and Classification of Random Aperiodic Signals, MS Thesis. Wright Patterson Air Force Base, Ohio: Air Force Institute of Technology, March 1971 (AD722647).

15. Intel Corporation. INTERP80 Manual, Santa Clara, CA, 1975.

16. Intel Corporation. Assembly Language Programming Manual, (98-004), Santa Clara, CA, 1975.

17. Intel Corporation. MDS-800 Intellec Microcomputer Development System Operators Manual, (98-129A), Santa Clara, CA, 1975.

18. Intel Corporation. SBC 80/20 User's Manual, (98-338B), Santa Clara, CA, 1976.

19. Intel Corporation. User Program Library, Santa Clara, CA, 1977.

20. International Business Machines Corporation. Structured Programming Text, (SR20-7149-1), Poughkeepsie, N.Y., 1975.

21. Jain, A.K. and W. G. Waller, On the Optimal Number of Features in the Classification of Gaussian Data, Technical Report 76-11936, East Lansing, Michigan: Michigan State University, August 1977.

22. Kabrisky, Mathew. A Proposed Model for Visual Information Processing in the Human Brain, Urbana, Illinois: University of Illinois Press, 1966.

23. Kanal, Laveen. "Patterns in Pattern Recognition: 1968-1974." IEEE Transactions on Information Theory, IT-20: 697-722 (November 1974).

24. Kulchak, Delbert. Target Classification by Time Domain Analysis of Radar Signatures. MS Thesis Wright Patterson Air Force Base, Ohio: Air Force Institute of Technology, December 1977.

25. Leary, J. R., with D. Hanson, R. Srba, E. Seward and R. Taylor. SOI User's Manual, Space Defense Center COSMOS Program Documentation ADC-COS-55-4-6, Colorado Springs, Co.: Aerospace Defense Command Operations, November 1974.

26. Motorola Corporation, M6800 Exorcisor User's Guide (MEX6800), Phoenix, Arizona, 1975.

27. McNichols, C. W. Applied Multivariate Data Analysis, Course Notes, Wright Patterson AFB Ohio: Air Force Institute of Technology, August 1978.

28. Michael, M. and W. C. Lin. "Experimental Study of Information Measure and Inter-Intra Class Distance Ratios on Feature Selection and Orderings," IEEE Transactions on Systems, Man and Cybernetics, SMC-3, No. 2, March 1973, p. 172.

29. Namin, P. J. "IFFN, A Technological Challenge for the '80s," Air University Review, Vol. 28, No. 6, September 1977.

30. Nie, N. H. et al. Statistical Package for the Social Sciences, New York: McGraw Hill Book Co., 1975.

31. Olson, E. A. Investigation of Feature Selection Criteria for Pattern Recognition Models Including the Fourier Transform, MS Thesis, Wright Patterson AFB, Ohio: Air Force Institute of Technology, March 1973 (AD760762).

32. Pacheco, N. S. Technical Review of a Pattern Recognition Program for Satellite Identification, USAF Academy Col., May 1974.

33. Sponaugle, T. Electrical Engineering Student (personal communication) Wright Patterson AFB, Ohio: Air Force Institute of Technology, February 1978.

34. Stearns, Stephen D. "On Selecting Features for Pattern Classifiers," Proceedings of the Third International Joint Conference on Pattern Recognition, 71-75. Coronado, California, IEEE, November 1976.

35. Tallman, O. H. The Classification of Visual Images by Spatial Filtering. PHD Dissertation, Wright Patterson AFB, Ohio: Air Force Institute of Technology, June 1969 (AD858866).

36. Wirth, Niklaus. "Program Development By Stepwise Refinement," Communications of the Association for Computing Machines, Vol. 14, No. 4, p. 221 (April 1971).

VITA

John R. Leary was born on 22 March 1943 in Tulsa, Oklahoma. In June of 1965 he graduated from the College of the Holy Cross in Worcester, Massachusetts, receiving a Bachelor of Arts degree with a major in mathematics. After working for two years with the IBM Corporation in Hartford, Connecticut, as an Associate Systems Engineer, he entered active duty and received his commission through OTS in September of 1967. He served as a Space Systems Analyst at two SPACETRACK radar sites, and as a Computer Systems Design Engineer, Section Supervisor, and Branch Chief at the NORAD Cheyenne Mountain Complex. In 1974, he received the Air Force Association Citation of Honor for achievement as a computer systems engineer. He has also received the Joint Service Commendation Medal for specific achievement as a systems engineer. In June of 1977, after completing a program of studies in post-graduate engineering science, he entered the School of Engineering at the Air Force Institute of Technology.

Permanent Address: 13 Wood Ct.
Terryville, Conn. 06786

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/78-12	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* A DEVELOPMENT SYSTEM FOR MICROPROCESSOR BASED PATTERN RECOGNIZERS		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John R. Leary Captain		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December 1978 13. NUMBER OF PAGES ~~370~~
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

Approved for public release;distribution unlimited
IAW AFR 190-17

JOSEPH P. HIPPS, Major, USAF
Director of Information 1-23-79

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Microprocessor programs
Pattern Recognition
Feature Selection

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A tool for developing microprocessor based pattern recognizers is presented. A two segment system of programs is implemented. One segment is a subsystem consisting of a generalized pattern classifier program and utility routines for an INTEL SBC 80/20 microprocessor system. The other segment is a subsystem of four interactive programs. These four programs support feature selection, pattern class definition and performance evaluation using procedures fitted to the classifier algorithm. This subsystem operates on a → next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

user supplied file of feature vectors. It produces a class defining structure for use by the classifier. It can use a TEKTRONIX 4014 for graphics support and will operate interactively within the CDC 6600 Intercom partition. Structured design, modular code, buffer allocation algorithms, and ANSI standard FORTRAN code make this segment transportable. The classifier segment requires an 8080 system. Less than 256 bytes of ROM are used. Data buffer locations and sizes, the number of classes and the number of features are specified by the user. Experiments produced estimates of classifier performance for this system. An error rate of less than ten percent is reported for one 26 class character recognition experiment.